WestVirginiaUniversity
THE RESEARCH REPOSITORY @ WVU

Graduate Theses, Dissertations, and Problem Reports

2005

Validating a neural network-based online adaptive system

Yan Liu
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation

Liu, Yan, "Validating a neural network-based online adaptive system" (2005). *Graduate Theses, Dissertations, and Problem Reports*. 4169.
https://researchrepository.wvu.edu/etd/4169

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Validating A Neural Network-based Online Adaptive System

Yan Liu

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer and Information Science

Bojan Cukic, Ph.D., Chair
Katerina Goseva-Popstojanova, Ph.D.
James Harner, Ph.D.
Ali Mili, Ph.D.
Arun Ross, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2005

Keywords: Validation, Online Adaptive System, Neural Network, Intelligent Flight Control, Support Vector Data Description, Validity Index.
Copyright 2005 Yan Liu

ABSTRACT

Validating A Neural Network-based Online Adaptive System

by Yan Liu

Neural networks are popular models used for online adaptation to accommodate system faults and recuperate against environmental changes in real-time automation and control applications. However, the adaptivity limits the applicability of conventional verification and validation (V&V) techniques to such systems. We investigated the V&V of neural network-based online adaptive systems and developed a novel validation approach consisting of two important methods. 1) An independent novelty detector at the system input layer detects failure conditions and tracks abnormal events/data that may cause unstable learning behavior. 2) At the system output layer, we perform a validity check on the network predictions to validate its accommodation performance.

Our research focuses on the Intelligent Flight Control System (IFCS) for NASA F-15 aircraft as an example of online adaptive control application. We utilized Support Vector Data Description (SVDD), a one-class classifier to examine the data entering the adaptive component and detect potential failures. We developed a "decompose and combine" strategy to drastically reduce its computational cost, from $O(n^3)$ down to $O(n^{\frac{3}{2}} \log n)$ such that the novelty detector becomes feasible in real-time.

We define a confidence measure, the validity index, to validate the predictions of the Dynamic Cell Structure (DCS) network in IFCS. The statistical information is collected during adaptation. The validity index is computed to reflect the trustworthiness associated with each neural network output. The computation of validity index in DCS is straightforward and efficient.

Through experimentation with IFCS, we demonstrate that: 1) the SVDD tool detects system failures accurately and provides validation inferences in a real-time manner; 2) the validity index effectively indicates poor fitting within regions characterized by sparse data and/or inadequate learning. The developed methods can be integrated with available online monitoring tools and further generalized to complete a promising validation framework for neural network based online adaptive systems.

GRANT INFORMATION

This work was funded in part by grants to West Virginia University Research Corp. from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV &V) Facility, Fairmont, West Virginia, NASA research grant No. NCC5-685; and from the NSF CAREER award CCR-0093315.

Acknowledgments

First of all, I would like to express my deepest sense of gratitude to my advisor Dr. Bojan Cukic for his patient guidance, encouragement and excellent advice throughout this study. As my mentor, he has not only guided me through the research, but also inspired me with his creativity and scientific approaches. Without his generous support and colossal help, I would not have accomplished this work.

I would like to express my appreciation to Dr. Ali Mili, Dr. Arun Ross, Dr. Katerina Goseva-Popstojanova and Dr. James Harner, who has shared his/her expertise and knowledge with me and offered professional advice throughout the course of my research. Without their help, I could not have gone through my Ph.D. study within the expected time.

I would like to thank the faculty and staff in our department. They create such a nurturing atmosphere that I enjoyed every day of studying and working here.

I would like to thank my lab colleagues, Paola Bracchi, Dejan Desovski, Vijai Gandikota, Yue Jiang, Lan Guo, Nick Bartlow, Martin Mladenovski, Nevena Samoska, Petar Popic, and Sampath Yerramalla for all their help, support and friendship. I also want to express my appreciation to my fellow researcher, Srikanth Gururajan, for his help and cooperation in my Ph.D. projects.

Last but not least, I would like to thank my parents and all my friends for always being supportive for me. Without their love and patience, I could not have gone this far.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

By definition, an adaptive system is a system that "*has a means of monitoring its own performance, a means of varying its own parameters, and uses closed-loop action to improve its performance* [1]." The functionality of an adaptive system evolves over time due to environmental changes. If learning and adaptation are allowed to occur after the system is deployed, the system is called online adaptive system [2].

In recent years, the use of biologically inspired soft computing models such as neural networks for online adaptation to accommodate system faults and recuperate against environmental changes has revolutionized the operation of real-time automation and control applications. Some of those online adaptive systems are safety critical, which require a reliable system performance under a real-time constraint. While online adaptive systems have the advantage of adaptability, they cause problems in terms of Verification and Validation (V&V). A vital reason is that an online adaptive system evolves over time and thus the validation techniques applied before its online adaptation are no longer applicable for its current stage.

Neural network models have been widely used as learning paradigms in online adaptive systems. The neural network plays an essential role as it adapts to the changes and accommodates the system faults. In the case of flight control applications, these system faults include failure mode conditions, which requires prompt reactions. As an online adaptive component, the neural network is expected to adapt to such failure conditions fast enough to provide accommodation in real-time. Furthermore, the neural network is used to predict certain parameters such as control derivative corrections to recuperate against system faults. However, the neural network's learning behavior is subject to the learning data and unreliable predictions are very likely to occur at poorly fitted regions. It is possible that abrupt environmental changes or unforeseen failure conditions beyond the learned domain

will cause poor prediction performance. It challenges the use of a neural network model in online adaptive systems and poses a serious problem in terms of V&V.

Among unconventional software systems where artificial intelligence is involved, applications such as flight control and robotic arms require adaptation because of the evolution of the environment. For such systems, it is impossible to build a static model that is able to provide reliable performance, especially under unforeseen conditions. The popular approach to employ such systems is to develop an off-line model, usually a pre-trained model for known functional regions. Then an online adaptive model is constructed in order to realize the adaptation in unknown functional regions. As an emerging application of online adaptive systems, the Adaptive Flight Control is one of the most promising real-time automation and control applications. The system achieves adaptability through judicious online learning, aids the adaptive controller to recover from operational damage (sensor/actuator failure, changed aircraft dynamics: broken aileron or stabilator, etc.). Some of these conditions are severe enough to be considered "failure" mode conditions that significantly affect system performance. National Aeronautics and Space Administration (NASA) conducted series of experiments evaluating adaptive computational paradigms (neural networks, AI planners) for providing fault tolerance capabilities in flight control systems following sensor and/or actuator failures. Experimental success suggests significant potential for further development and deployment of adaptive controllers [3, 4]. Nevertheless, the (in)ability to provide a theoretically sound and practical verification and validation method remains one of the critical factors limiting wider use of *"intelligent"* flight controllers [5, 6, 7].

Adaptive systems in general are considered inherently difficult to validate. System uncertainties coupled with real-time constraints make traditional V&V techniques useless for online adaptive systems. Development and implementation of a non-conventional V&V technique is a challenging task. After a thorough examination using formal methods on some learning paradigms [2], we gained an in-depth understanding of such systems and discovered that environmental changes (learning data) have a significant impact on system behavior. For a safety-critical system such as a flight control application, these changes must be observed, detected and well-understood before system deployment. Further, the adaptation caused by such changes must be monitored in real-time and the consequences of the adaptation must be measured and controlled.

Figure 1.1 illustrates a typical adaptive flight control application. Our research focuses on these two layers of the online adaptive component, referred to as the input layer and the output layer in Figure 1.1. At the input layer, environmental changes are captured through sensor readings. These independent values are coupled with discrepancies between the re-

sponse of a pre-planted physical process and the output of an off-line reference model as they react to the current environmental conditions. Together, they are fed into the adaptive component as learning data. The learner adapts to the learning data and may accommodate certain failure conditions. It is possible that abrupt/abnormal environmental changes, especially severe failure conditions, cause dramatic adaptation behavior and transient unreliable performance of the learner. As the learner learns and adapts to the environmental changes, it also produces certain parameter values as corrections in response to the changes. Thus, at the output layer, the learner recalls what it has learned and generates derivative corrections as compensation to the off-line model output. The corrected output are then sent to the next component (e.g. an actuator/ a controller) for further actions. The validity and correctness of such corrections are crucial to system safety.

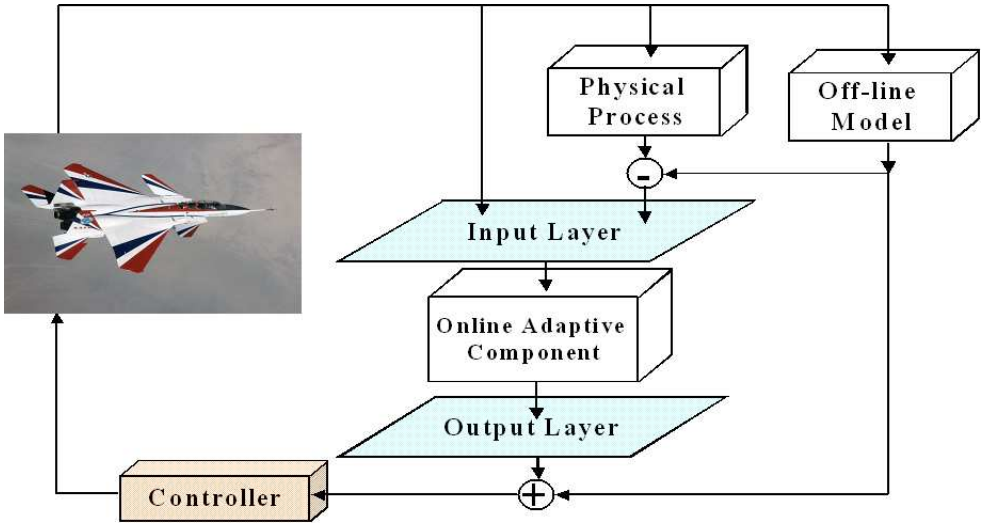


Figure 1.1: The input layer and output layer of a typical adaptive flight control application.

At the input layer of an online adaptive system, the failure-prone data need to be analyzed and potential failures detected. When a failure occurs, the learner learns the failure conditions and adapts to the corresponding environmental changes. At the output layer, the system safety relies greatly on the accommodation performance of the learner. The reliability of the predictions made by the learner depends on how well and how fast the learner adapts to failure conditions. In order to evaluate the accommodation performance of the learner and validate its prediction performance, the confidence level must be measured as an indication of trustworthiness and reliability of the system output.

We seek for validation methods at both layers. Our investigation suggests a need for near real-time failure detection at the system input layer and a feasible validation approach

to validate the prediction performance at the system output layer. Thus, we propose a novel validation approach that consists of two methods to perform validation at these two layers. The failure detection at the input layer relies on robust novelty detection techniques. Such techniques are used to examine the learning data on which the adaptive component is trained and detect the failures efficiently and accurately. The validation at the output layer is performed after the adaptation to verify the correctness of the output and evaluate the accommodation abilities of the learner.

Accordingly, we developed two validation methods. At the input layer, a real-time failure detection technique is implemented before the data can propagate into the learner and cause adaptation. As an independent novelty detector, it detects failures based on the knowledge collected from the nominal system behavior only. When failure occurs, the adaptive learner accommodates the failure in real-time and predicts certain parameter values as compensations/corrections to the off-line model output in response to the failure conditions. Hence, at the output layer, an online reliability measure associated with each prediction produced by the learner is generated for validity check. At last, both approaches serve as two major components of a validation framework that can be generalized to extensive online neural network-based adaptive control systems.

In this chapter, we introduce the framework in which the research is conducted and give a brief description of the specific system.

1.1 The Intelligent Flight Control System

As an example of online adaptive systems, also the major application of our research, the Intelligent Flight Control System (IFCS) was developed by NASA as "*a revolutionary flight control system that can efficiently identify aircraft stability and control characteristics using neural networks and use this information to optimize aircraft performance in both normal and simulated failure conditions* [5]."

The diagram of Figure 1.2 illustrates the architecture of the IFCS. The purpose the online neural network fulfills within the IFCS can be described as follows. As an online function approximator, the online neural network is expected to adapt to system faults and accommodate such faults. When needed, it predicts the differences between the parameter estimation of the stability and control derivatives by a Parameter Identification (PID) tool and the baseline derivatives generated by the Pre-Trained Neural Network (PTNN, as the Baseline Neural Network in Figure 1.2). In essence, the system adapts the aircraft model in response to changes in the aircraft or model inaccuracies and tries to accommodate such

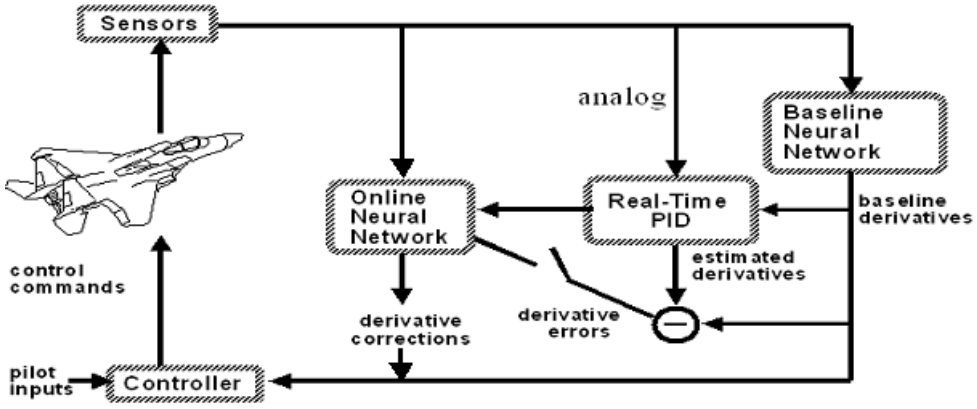


Figure 1.2: The Intelligent Flight Control System.

changes and potential failures.

Within the IFCS, the model updates are to occur in real-time during flight. The model must be capable of providing aerodynamic derivatives to the aircraft controller at least 10 times per second. Therefore, the online neural network learns the discrepancies between the baseline neural network and the real-time parameter identification (PID) along with the sensor data that it fetches from the data buffer filled by the PID with accumulated analog parameters. It also produces parameters to be used by the controller to optimize the flight response of the aircraft under a variety of maneuvering conditions. In either scenario, the online learning should provide reliable performance for further mapping of those key control parameters.

Within an online adaptive flight control application, the neural networks are specifically designed for online learning purposes, where the ability of prompt adaptation is highly desired. In IFCS, the neural network models chosen as online adaptive learners are the Dynamic Cell Structures (DCS). As an extension of a Self-Organizing Maps, the DCS network dynamically adjusts its own structure to map the learning data. The structural flexibility of DCS offers the ability of prompt adaptation in response to environmental changes and system faults. The DCS network has two operational modes, the learning mode and the recall mode.

- When a DCS network is in learning mode, it is trained on the flight data. The DCS network adjusts its structure to adapt to the current operational domain and tries to

accommodate failure conditions in a timely manner.

- In the recall mode, a DCS network is used for prediction. After adaptation, as a mapping function, the DCS network is then employed to recall parameter values at any chosen dimension.

The DCS network will be further discussed in Chapter 5.

1.2 Research Contributions

We developed a validation methodology suitable for neural network-based online adaptive systems. We applied our approach to IFCS that employs DCS network as the online learning paradigm. Presented V&V methodology consists of two online validation techniques.

At the input layer, we utilized a machine learning tool, Support Vector Data Description (SVDD), to examine the data entering the adaptive component and detect unforeseen and potentially hazardous patterns considered failure conditions. In order to meet the real-time requirement of IFCS, we developed a "decompose and combine" strategy to drastically reduce the computational cost, from $O(n^3)$ down to $O(n^{\frac{3}{2}} \log n)$. Our simulations indicate that this transformation reduces the run time of SVDD by 90%, whereby the failure detection becomes computationally sufficient for the IFCS domain. The developed technique effectively detects system failures and provides validation inferences in a real-time manner.

At the output layer, we developed another validation method for validating the neural network outputs of the IFCS. We define a reliability-like measure, the validity index, for the Dynamic Cell Structure (DCS) network used for adaptive learning in IFCS. The validity index reflects the confidence level associated with each output and thus can be used to perform validity checks and verify the accommodation performance of the online neural network.

Experimental results suggest that both techniques can be integrated with available online monitoring tools and further generalized to complete the validation framework for neural network-based online adaptive systems. The online monitoring tools implemented by Yerramalla et. al. [8, 9, 10] detect unstable (unusual) adaptive behavior of the online adaptive learning using Lyapunov theory.

1.3 Dissertation Structure

In this chapter we introduced the V&V problem of neural network-based online adaptive systems. Chapter 2 discusses the related work in V&V of online adaptive systems, V&V of neural networks, and popular novelty detection methods. Chapter 3 describes a validation framework for validating the IFCS as a typical example of neural network-based online adaptive systems. Chapter 4 describes the novelty detection tools for failure detection at the system input layer and Chapter 5 presents the validity index in DCS network for validity checks at the system output layer. Chapter 6 presents the experimental results with IFCS including the demonstration of real-time failure detection and validity index in DCS. In Chapter 7, we summarize our research efforts and discuss future work.

Chapter 2

Related Work

There have been a number of V&V techniques for intelligent systems developed in recent years with the increasing attention caused by the popular use of such systems. In this chapter we will summarize some of those techniques from both traditional software V&V and novel V&V approaches to neural network-based adaptive systems. Moreover, we present our investigation of related research in novelty detection techniques.

2.1 Verifying Online Adaptive Systems

Considering the methods for software product verification, there are three major families usually mentioned in the literatures [11, 12]:

- **Fault Avoidance methods.** They are based on the premise that we can derive systems that are fault-free by design.
- **Fault Removal methods.** They are based on the premise that we can remove faults from systems after design and implementation.
- **Fault Tolerance methods.** They are based on the premise that we can take measures to ensure that residual faults do not cause failure.

Unfortunately, neither of these three methods is applicable as-is to adaptive systems, for the following reasons:

- **Fault Avoidance.** Formal design methods [13, 14] are based on the premise that we can determine the functional properties of a system by the way we design it and

implement it. While this holds for traditional systems, it does not hold for adaptive systems, since their design determines how they learn, but not what they will learn. In other words, the function computed by an online adaptive system depends not only on how the system is designed, but also on what data it has learned from.

- **Fault Removal: Verification.** Formal verification methods [15, 16, 17] are all based on the premise that we can infer functional properties of a software product from an analysis of its source text. While this holds for traditional systems, it does not hold for adaptive systems, whose behavior is also determined by their learning history.

- **Fault Removal: Testing.** All testing techniques [18, 19, 20] are based on the premise that the systems of interest will duplicate under field usage the behavior that they have exhibited under test. While this is true for traditional deterministic systems, it is untrue for adaptive systems, since the behavior of these systems evolves over time. We have observed in [12] that adaptive systems fail to meet this requirement (of maintaining or enhancing their behavior) even when they converge.

- **Fault Tolerance.** Fault tolerance techniques [21, 22, 23, 24, 25] are based on the premise that we have clear expectations about the functions of programs and programs parts, and use these expectations to design error detection and error recovery capabilities. With adaptive systems, it is not possible to formulate such expectations because the functions of programs/ program parts are not known at design time.

As an emerging intelligent paradigm, the online adaptive system becomes more and more popular. Yet, the research effort on V&V of such systems is still rare. Fortunately, many researchers start to turn their attention to the V&V problem. Among most recent research, some effort has been dedicated to analyzing certain properties of online adaptive learning in general. Others focus on methods that can be applied in an online fashion to assure the performance. Most proposed approaches address the significant impact of learning data on system behavior and investigate certain properties of learning in order to ensure system safety through online monitoring or rule checking. We summarize the existing approaches into two categories, namely, static V&V approaches and dynamic V&V approaches.

2.1.1 Static V&V Approaches

Most static verification and validation methods focus on the inherent properties of online adaptive learning. These methods are used to theoretically establish the correctness of the

learning behavior with respect to the requirement specifications. Researchers often employ mathematically rigorous theories to prove functional properties and/or operational properties.

Approximation theory is used to prove certain families of neural networks as universal approximators. Examples are Multi-Layer Perceptron (MLP) networks and Radial Basis Function networks(RBF). These two types of neural networks are very popular choices for adaptive learners and have been proven to be universal approximators [26]. It is claimed that an MLP (or RBF) network with sufficiently large number of neurons can approximate any real multi-variate continuous function on a compact set. However, the number of neurons has to be pre-defined before the system deployment. Usually, the number of neurons an MLP/RBF network requires to map a complex function may have to be very large. In the instance of an online adaptive system, the proven theory offers little guidance in validating the online learning performance of a neural network based adaptive learner.

There are several properties that are of special interests to be analyzed. Based on predicate calculus, our previous research defines two important properties of an online learning system as follows [2].

- Monotonicity. A learning system that has monotonicity is ensured to learn in a monotonic fashion, so that whatever claims we can make about the behavior of the system prior to its deployment are upheld while the system evolves through learning.
- Safety. It ensures that as the adaptive system evolves through learning, it maintains some minimal safety property.

Based on monotonicity and safety, there are two venues for verifying adaptive systems: one based on monotonic learning (the adaptive equivalent of testing), and one based on safe learning (the adaptive equivalent of proving). However, using refinement-based logic to search for sufficient conditions for monotonic learning and safe learning is nearly impossible because of the high complexity and nonlinearity of the neural network system.

Empirical methods are also available for testing and verifying the adaptive learner against certain safety and reliability requirements [27]. The train-test-retrain scheme for validating neural network performance is a popular V&V approach. Yet, this time-consuming procedure is not suitable for an online adaptive system due to the fact that the network has to learn in near real-time. The bias-variance analysis provides guidelines on the generalization performance of a learning model, but can hardly be applied to improve the prediction performance of an online learning system.

The static verification methods using formal methods and approximation theories provide an insightful analysis on neural network based adaptive systems. For most neural computing systems, empirical methods are practical for performance validation. However, there is a widespread agreement that such static approaches are inapplicable to online adaptive systems, whose function evolves over time and responds differently to various environmental changes.

2.1.2 Dynamic V&V Approaches

Instead of statically validating the learning properties of an adaptive system, most dynamic approaches adopt the online monitoring scheme to cope with the evolving performance of an online adaptive learner. These methods concentrate on three different aspects(phases) of an online adaptive system.

1. For any learning system, training data is always gathered before the learner is used for prediction. Verification of the training data includes the analysis of its appropriateness and comprehensiveness. The strong emphasis on domain specific knowledge, its formal representation and mathematical analysis is suggested in [28]. Del Gobbo and Cukic propose the analysis of the neural network with respect to conditions implying the existence of the solution (for function approximation) and the reachability of the solution from any possible initial state. Their third condition can be interpreted as condition for preservation of the learned information. This step is not fully applicable to on-line learning applications since training data are related to the real-time evolution of the system state, rather than the design choice. However, as proven by our previous investigation using formal methods [2], the training data has a very significant impact on system behavior. In a safety-critical system, the ability of "novelty detection" is crucial to system safety. It helps to detect suspicious learning data that is potentially hazardous to the system operation.

2. Online monitoring techniques have been proposed to validate the learning process. In a recent survey of methods for validating online learning neural networks, O. Raz [27] acknowledges the online monitoring techniques as a significant potential tool for the future use. Another promising research direction, according to Raz, is periodic rule extraction from an online neural network and partial (incremental) re-verification of these rules using symbolic model checking. In [29], Taylor et.al. focus their effort on the Dynamic Cell Structure. They propose a prototype for real-time rule extraction

in order to verify the correctness of DCS learning performance. In [30], M. Darrah et. al. present rule extraction from DCS network learning and suggest future examination of performance based on such rules. Practical hurdles associated with this approach include determining the frequency of rule extraction and impracticality of near real-time model checking of complex systems.

As a parallel research, S. Yerramalla et. al. develop a monitoring technique for the DCS neural network embedded in the IFCS [8, 31] based on Lyapunov stability theory. The online monitors operate in parallel to the neural network with the goal of determining whether (or not), under given conditions, the neural network is convergent, meaning that all state transition trajectories converge to a stationary state. The online monitor is theoretically founded and supported by an investigation of mathematical stability proofs that can define the engagement (or disengagement) of the online monitor.

- It is noticed that a few efforts have been made towards the validation of prediction performance, where the system is in operation after learning for a certain period of time. One related research done by Schumann on Sigma-Pi networks uses a Bayesian approach to monitor the prediction performance for an online adaptive system in a real-time manner [32]. In some cases, neural networks are modified to provide support for testing based (or online) validation of prediction performance. For example, Leonard et. al. [33] suggest a new architecture called Validity Index Net. A Validity Index network is a derivative of Radial Basis Function (RBF) network with the additional ability to calculate confidence intervals for its predictions based on the probability density of the "similar" training data observed in the past.

2.2 Traditional V&V Techniques on Neural Networks

Because online learning systems are most often used in life-critical (e.g. flight control) and mission-critical (e.g. space) applications, they are subject to strict certification standards, leaving a wide technological gap between the requirements of the application domain and the capabilities of available technologies; the goal of our research is to narrow this gap. Hence, we survey existing approaches to V&V of neural networks.

Traditional literature describes adaptive computational paradigms, neural networks in particular, with respect to their use, as function approximators or data classification tools. Validation on these systems is usually based on a train-test-re-train empirical procedure. Some bibliographic references also propose methods as part of the training algorithm of

neural networks for validation [34, 5]. The ability of interpolating and/or extrapolating between known function values is measured by certain parameters through testing. This evaluation paradigm can be reasonably effective only for pre-trained adaptive systems, which does not require online learning and adaptation and remain unchanged in use. In [35], Fu interprets the verification of a neural network to refer to its correctness and interprets the validation to refer to its accuracy and efficiency. He establishes correctness by analyzing the process of designing the neural network, rather than the functional properties of the final product. Gerald Peterson presents another similar approach in [36] by discussing the software development process of a neural network. He describes the opportunities for verification and validation of neural networks in terms of the activities in their development life cycle, as shown in Figure 2.1.

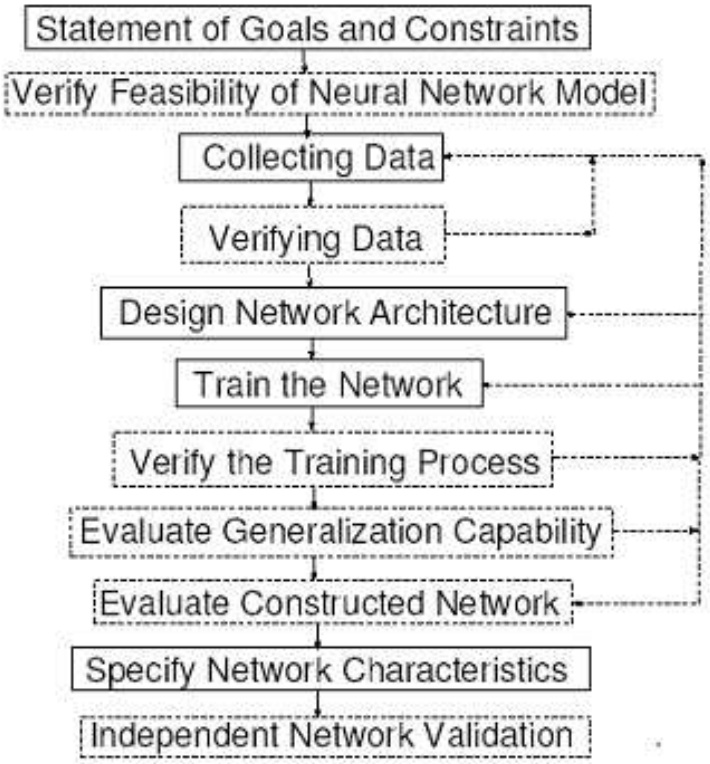


Figure 2.1: The development cycle of a neural network.

As we can see from Figure 2.1, there is a focus on V&V of adaptive systems based on the training data. Verification of the training data includes the analysis of appropriateness and comprehensiveness. However, in online learning mode, this technique may not be appropriate

due to its real-time training. The data is collected in such a way that the training is completed under intensive computational requirements. An applicable approach for verifying the data is novelty detection, which is presented in Section 2.3.

Verification of the training process typically examines the convergence properties of the learning algorithm, which is usually pre-defined by some criteria of error measure. In [37], K.J. Hunt et.al. investigate all different methods for error estimation techniques and make detail comparison among them. Nonetheless, effective evaluation methods of interpolation and extrapolation capabilities of the network and domain specific verification activities are still based on empirical testing [38]. Literature addressing the problem analytically are very scarce. In the field of function approximation theory, MLP networks have been proven to be universal approximators for being able to achieve any given accuracy provided a sufficient number of hidden neurons [26]. The mathematical analysis and proof can be seen as another effort for validating the learning process whereas it can provide theoretical proof for their capabilities of function approximation. The weakness of such analytical proof is that it remains impractical for online adaptive learning systems as the system function evolves.

Most recently proposed techniques on V &V of neural networks are based on empirical evaluation through simulation and/or experimental testing. There are also other approaches to V &V of dynamic neural networks. In an attempt to solve the dilemma of plasticity and stability for neural networks, S. Grossberg [39, 40] derives a new paradigm, referred to as the Adaptive Resonance Theory (ART-1/2/3). Within such a network, there are two components charging seen and unseen data respectively. As interesting as is, it provides better understanding for our problem other than applicable tools for validation and verification. Another new architecture which can be extended for our research goal is the Validity Index network presented by Leonard et. al. in [33]. The validity index in a Radial Basis Function neural network is a confidence interval associated with each network prediction for a given input. It will be further discussed in Chapter 5.

2.3 Novelty Detection

Novelty detection is defined as "the identification of new or unknown data or signal that a machine learning system is not aware of during learning. [41, 42]" Novelty detection is considered an important approach for validating adaptive systems using neural networks as adaptive models [43, 44]. Usually when "novel data", which are radically different from seen data in the phase of training and testing of a neural network model, arise in the new input data after the testing, there is a great possibility of system malfunctioning. By using novelty

detection, we are be able to discover such "novel data" and its erroneous classification as well, and thus appropriate warnings can be posted and proper actions can be taken to prevent abrupt loss of system functionality.

In general, novelty detection is often used as an indicator of potential erroneous consequences of outliers and hence system emergency. In particular, for flight control systems, not only all possible failure mode data are very hard to collect, but also all possible normal mode data are impossible to be completely collected. Therefore, novelty detection can be potentially beneficial for identifying abnormal inputs based on its capability of predicting the difference between safe data and novel data.

During the past decades, many researchers have investigated this technique and a number of approaches have been developed [44, 45, 46, 47, 48, 49, 50]. In principle, there are two categories of approaches. One is the statistical approach, which utilizes certain statistical models for density estimations or clustering [41]. The other is the machine learning approach, which uses neural networks or other connection models to predict novelty for a specific domain [42]. The "novelty" defined in statistical models is based on the unconditional probability density function of data, while in the machine learning approaches, it is defined based on the pre-defined classification of the data and the learning performance of such models.

2.3.1 Statistical Models

Statistical models for novelty detection can be further classified into parametric models and non-parametric models [41]. Parametric models assume that the data fits a certain family of distribution, for example, Gaussian (normal) distribution, Poisson distribution, etc.. Such models are used to search for the "optimal" parameter values for these distributions. However, very little information regarding the underlying data distribution is available for most real-world applications. Therefore, due to its requirement of extensive knowledge of a data domain to build a parametric model, parametric models are not as applicable and flexible as non-parametric models in many domains. Unlike parametric models, non-parametric models do not make any assumptions concerning the form of data distribution. In most cases, non-parametric models outperform parametric models with greater flexibility and independence. Nonetheless, nonparametric models are not necessarily a better choice when computational complexity has the highest priority because most of these models have relatively high computational cost. In the rest of this section, we will discuss some popular parametric models as well as some nonparametric models.

1. Parametric Models

   The most popular and well-studied model is the Gaussian Mixture model (GMM). GMMs are among the most advanced and sophisticated statistical methods for density estimation. When the data are unlabeled, the GM modeling will lead to the identification of groupings (clusters) among the data, for further classification and/or measurement. The Gaussian mixture model has the following form:

   $$f(x) = \sum_{i=1}^{N} \alpha_i \phi(x; \mu_i, \Sigma_i);$$

   where
   α_i's are mixing portions and $\Sigma_i \alpha_i = 1$. Each Gaussian distribution has a mean μ_i and covariance matrix Σ_i. Estimation of the parameters is usually carried out by maximum likelihood, using the Expectation Maximization (EM) algorithm. GMMs can be used as a powerful tool for data clustering analysis and density estimation. However, it suffers from the curse of dimensionality. In other words, when the dimensionality of the data is high, we will need a large amount of data for training the model.

   Another well-known approach is the Hidden Markov Models (HMMs). As a stochastic model for sequential data, an HMM consists of a finite set of (hidden) states. The probabilities associated with each state are determined by some certain probability distribution. Transitions among the states are defined by a set of probabilities, referred to as transition probabilities. In a particular state, an outcome or observation can be predicted based on the associated probability distribution. HMMs have been proven to be suitable models for intrusion detection applications [51]. Yet HMMs have few merit for solving the novelty detection problem for an adaptive system whose definition of states is much harder to obtain.

2. Nonparametric Models

   Among all available nonparametric models, popular and well-developed methods are Parzen windows and k-Nearest Neighbor (KNN) models.

   With only a few parameters, the Parzen window method is claimed to be straightforward and relatively effective. It employs a Gaussian kernel model for each pattern within the pattern vector to determine the centroid of the kernel. A global smoothing parameter for controlling the height of the kernel is provided to justify classification performance of the kernel. In Bishop's experiments [43], he selected the smoothing parameter as the average of distance of the ten nearest data points over all the training

data set. Furthermore, a threshold can be drawn to determine whether the data are within the scope of the kernel or outside the scope that could be claimed as novel data. Although this method is widely used because of its simplicity and effectiveness in some application domains, it holds a limitation that the width of the kernel is pre-defined and thus is fixed for all training data points through the whole process. Because of the limitation, this method sometimes may lead into false coverage of truth region and is unable to distinguish the novel data points.

The k-nearest model has been used for estimating entropy in data sets [52, 53]. Given a pre-defined k, by running the k-nearest neighbor algorithm, information about examples that have common properties is gained. Finally, if the test data lies beyond a certain Euclidean distance threshold from any neighborhood of those k-nearest regions, then it can be labeled as an outlier. This method has a drawback that the number k is needed before actually clustering the data. In the meanwhile, by using Euclidean distance measurement, it makes an implicit hypothesis concerning the data distribution, which attributes too strong assumptions for some real-world data.

2.3.2 Machine Learning Tools

Among popular machine learning approaches, inductive learning learns from examples and generalizes based on learned knowledge. There exist two types of inductive learning tools that are considered favorable models for novelty detection. They are conventional neural networks and data mining tools. Conventional neural networks such as MLP and RBF, Self Organizing Maps (SOM) prevail in the field mainly because for such models no a prior knowledge is required for the domain [42]. In particular, SOM are popular models for clustering. However, costly computation and overfitting are significant problems associated with those models. They usually require massive computational effort and hence intensified real-time calculation for the purpose of online monitoring makes these models nearly impractical. Therefore, we are more inclined to adopt learning techniques that are more computationally efficient.

As a subclass of inductive learning, concept learning takes examples of a concept and tries to form a general description of the concept through learning. The examples are usually described using attribute-value pairs. Data mining techniques are popular approaches used to obtain the description. After that, the description can be employed for novelty detection. Among such methods, well-known data clustering techniques such as k-means clustering can be very effective if empirical testings are allowed for the data domain to decide the optimal parameters before its actual deployment. Although impractical, these methods can still be

considered for off-line testing to provide domain knowledge for comparison purposes (to be compared with other data mining tools).

As an emerging new tool for classification, the Support Vector Machine (SVM) is a statistical learning paradigm [54, 55]. Most real-world classification problems face the difficulty caused by non-separable nonlinear data space. Other than designing complex computational model to approximate the separation, Vapnik derived the support vector machine based on the idea of mapping the data into a higher dimensional space, which is called the feature space, whereby we can define a separating hyperplane [54]. The principal task of the SVM is to find the maximum margin hyperplane that maximizes the minimum distance from the hyperplane to the closest data points. The key concept of the SVM is that of VC (Vapnik Chervonenkis) dimension, the VC dimension of an hypothesis space is a measure of the number of different classifications implementable by functions from that hypothesis space. Theoretically, it is proven that the generalization ability of this learner depends on the VC dimension that the learning machine implements rather than on the dimensionality of the space. Evidently, with SVM, we are able to avoid the problem of locality vs. generability as well as the dramatically growing computational costs in high dimensional space.

Two key elements in the implementation of SVM are the techniques of mathematical programming and kernel functions. The parameters are found by solving a quadratic programming problem with linear equality and inequality constraints that can be reduced to a convex optimization problem. The flexibility of kernel functions allows the SVM to search a wide variety of hypothesis spaces, while at the same time, there arises the problem of choosing a fitting kernel function, where researchers have proposed many feasible solutions [56, 57, 58].

Based on the SVM theory, Tax et.al. derived the support vector data description (SVDD) method [59, 60, 61]. The SVDD method focuses on the one class classification problem. It originates from the idea of finding a sphere with the minimal volume to contain all data [62]. As shown in Figure 2.2, by forming a circle around most objects that we consider normal data, we obtain a "boundary" for labeling those objects outside the circle as "outliers", in another word, "novel data". The area inside the boundary can be seen as the "safe region" representing the characteristics defined by all those objects inside. Later on, by applying kernel functions, one-class classification problems in non-linear space can also be solved.

Other data mining approaches such as association rule learning, Bayesian techniques can also be used for novelty detection. In T. Yairi's paper [63], a sample application of applying association rule learning is presented. By monitoring the variance of the confidence of particular rules inferred from the association rule learning on training data, it provides

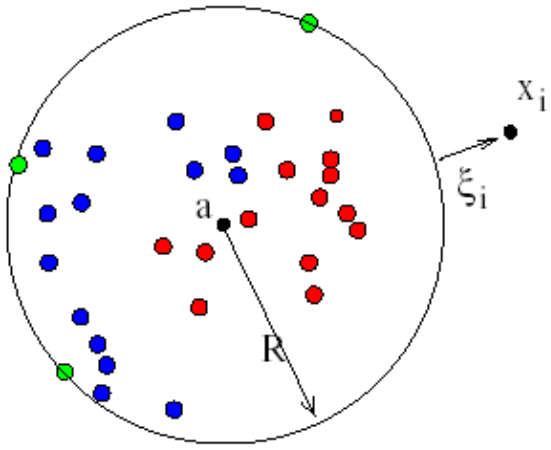


Figure 2.2: The hyper-sphere containing the "normal" data points, defined by the radius R and the center a. x_i is considered the outlier with $\xi_i > 0$ [62].

information on the difference of such parameters before and after the testing data entering the system. Hence, with some pre-defined threshold, abnormalities can be fairly detected. Our investigation in association rules starts with 16 benchmark data sets selected from the UC Irvine [64] machine learning data repository. The experimental results show that the association rules could be an effective tool for novelty detection if and only if there are sufficient rules made available by the data miners [65, 66]. However, the discretization of numeric values and how to justify and synthesize the results for applying hundreds of rules are two very challenging problems to which we cannot find a feasible solution.

2.3.3 Comparison of different methods

Being utilized for validating an online adaptive system, our desired method for novelty detection should be theoretically sound, practically robust and computationally efficient. Furthermore, for an online adaptive system, the IFCS in particular, failure data is extremely hard to collect and obtain. The number of different failure modes can be infinite. Hence, what we need is a one-class classification method. A one-class classification method assumes that *"only information of one of the classes, the target class, is available* [62]." Therefore, a one-class classification method only uses the target (in the case of novelty detection, "normal") data when the outliers ("novel" data) are absent. The objective of a one-class classifier is *"to define a boundary around the target class, such that it accepts as much of the target objects*

as possible, while it minimizes the chance of accepting outlier objects [62]."

As our literature review suggests, Parzen windows, GMM models, k-nn models, k-means clustering, SVDD and SOM networks are popular and sophisticated models that can be employed as one-class classification methods and thus worth further exploration. The ideal tool we need should be able to offer reliable performance and near real-time detection efficiency. In order to find the most suitable novelty detection method, we consider these six methods and compare their performance based on the following principles, summarized by Markou et.al. in [41].

- Principle of robustness and trade-off. *"A novelty detection method must be capable of robust performance on test data that maximizes the exclusion of novel samples while minimizing the exclusion of known samples. This trade-off should be, to a limited extent, predictable and under experimental control."*
- Principle of uniform data scaling. *"In order to assist novelty detection, it should be possible that all test data and training data after normalization lie within the same range."*
- Principle of parameter minimization. *"A novelty detection method should aim to minimize the number of parameters that are user set."*
- Principle of generalization. *"The system should be able to generalize without confusing generalized information as novel."*
- Principle of independence. *"The novelty detection method should be independent of the number of features, and classes available and it should show reasonable performance in the context of imbalanced data set, low number of samples, and noise."*
- Principle of adaptability. *"A system that recognizes novel samples during test should be able to use this information for retraining."*
- Principle of computational complexity. *"A number of novelty detection applications are online and therefore the computational complexity of a novelty detection mechanism should be as less as possible."*

According to the characteristics of our research problem, what we concern the most is the robustness, independence and computational efficiency.

We compare all the potential candidates for novelty detection. Taking an artificial data set, the highly non-linear "banana" data set, we apply six different methods to obtain the

Table 2.1: Novelty Detection Results

	Novelty present	Novelty absent
Test positive	True Positives TP	False Positives FP
Test negative	False Negatives FN	True Negatives TN

data description. These six methods are: Parzen windows, Gaussian mixture modeling, support vector data description, k-nearest neighbors, k-means clustering and Self Organizing Maps. As presented in Table 2.1, the measures of goodness of a method are based on the following statistical definitions:

- Sensitivity. This indicates how good a method is in terms of finding the true positives. It is represented by the True Positive Rate (TPR) (See Table 2.1). Sensitivity provides the proportion of true "novelties" picked out by the test.
- Specificity. This is the ability of the method to determine the ones which are true negatives. It is represented by the True Negative Rate (TNR) (See Table 2.1). Specificity provides the proportion of true "non-novelties" (normal data/event) decided by the test.

The false negative rate and true positive rate are in fact fractions. Thus, we have:

$$FNR + TPR = 1.$$

As a statistically well-known metric, a Receiver Operating Characteristic (ROC) curve is a graphical representation of the trade-off between the false negative rate and false positive rate as we vary the arbitrary threshold. The ROC curve can be viewed as the representation of the tradeoffs between sensitivity and specificity. Another commonly used measure is the Area Under the ROC Curve(AUC). It is a summary measure of the test accuracy which varies from 0.0 to 1.0. In its definition regarding a one-class classifier [62], the AUC can be perceived as the probability that a randomly selected "normal" event/data will be regarded with high suspicion (likelihood or rating or measurement) than a randomly considered "novel" event/data. Hence, the smaller the value of the AUC, the better classification performance is provided by the one-class classifier.

The performance of the selected six methods is presented by ROC curves and the AUC values in Figure 2.3 (b). Figure 2.3 (a) shows the data description results obtained from these six methods represented by different lines.

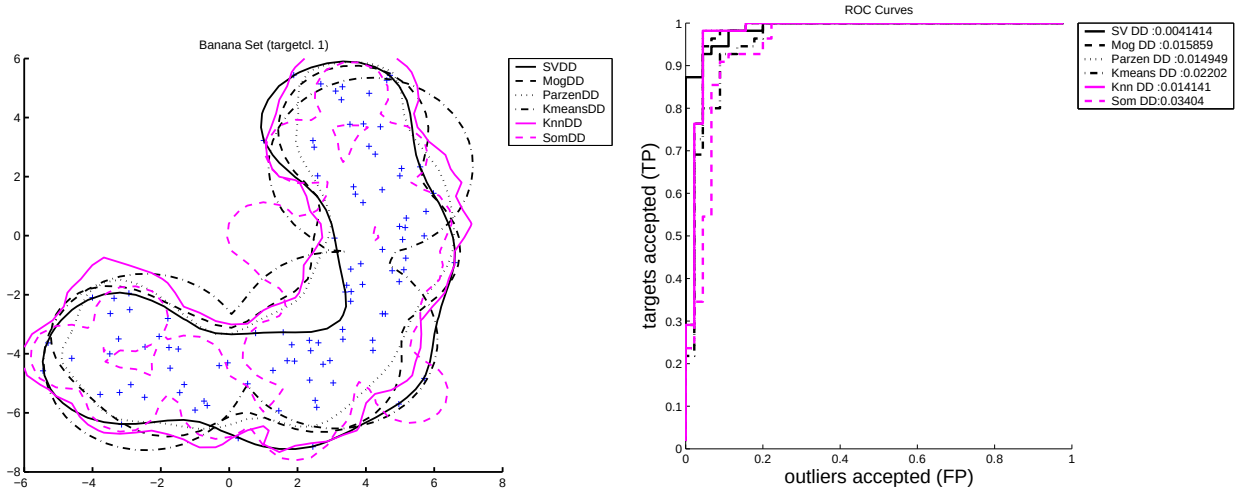


Figure 2.3: Compare six different methods on banana data set. (a): Data descriptions obtained from different methods. (b): ROC curves for different methods.

While comparing the performances of different models, we can conclude that the SVDD, GMMs(referred to as Mog in 2.3), k-Nearest Neighbors(kNN) and Parzen windows are the ones that provide better classification performance than others. Among all six methods, kNN method has the highest computational cost. It also requires an optimal number, k, to be selected before the classification. Although GMMs are considered parametric models, it can be viewed as semi-parametric since it uses very few parameters as well as fewer kernels than the Parzen window method to build a model. GMMs provide more flexibility and precision in modeling the underlying statistics of data than Parzen window method. The Gaussian mixture model can effectively detect novel data by evaluating the data based on the kernels and their combinations. However, this method needs sufficient amount of data to build a solid model. Consequently, the computational effort is much more expensive than other models. The smallest AUC value of SVDD (0.0041414) indicates the soundness of this method. The advantage of SVDD lies in the fact that it makes no assumption concerning the underlying distribution of the data and thus there is no need to make any probability density estimation.

2.4 Summary

Traditional V&V methods are inapplicable to online adaptive systems because of the adaptivity of such systems. Although they cannot offer much practical help in real-time validation of online adaptive systems, analytical approaches provide us with meaningful insight into the V&V problem of such systems. We focus our attention on dynamic methods such as novelty detection and online monitoring techniques that are considered practical for validating online adaptive systems.

As suggested by our research effort of V&V using formal methods [2], the learning data has a very significant impact on the adaptive behavior of an adaptive system. This motivates us to seek for a feasible and robust novelty detection approach to effectively and efficiently detect the failure conditions to better assure the adaptive learning performance. The online restrictions require the candidate method to be computational efficient and flexible. Based on our investigation in novelty detection methods, we select the support vector data description as our novelty detection technique to detect failures based on the nominal data only.

In the context of validating an online adaptive system at both input and output layers, we will investigate two methods. At the input layer of an online adaptive system, we expect to employ the SVDD tool to accurately detect failures in real-time. At the system output layer, we will verify the accommodation performance of the online adaptive learner by examining the output data generated by the learner after adaptation. The validity index can be interpreted as a reflection of trustworthiness of each individual output. We expect to provide the trustworthiness of the output by some confidence measure such as the "validity index" proposed by Leonard et.al. for Radial Basis Function (RBF) network.

Chapter 3

A Validation Framework

The major task of an online adaptive system is to adapt to system faults, especially failure conditions, and accommodate such faults. Within a safety-critical application such as the Intelligent Flight Control System (IFCS), the online neural network plays a very important role in assuring system safety and reliability. While they hold great technological promise, the online learning paradigms pose serious problems in terms of verification and validation, especially when viewed against the background of the tough verification standards that arise in their predominant application domains (flight control, mission control). Our investigation in V&V of adaptive systems shows that conventional V&V techniques are generally static and thus impractical for such systems because of the continual changes of the environment. Thus, we propose a dynamic validation approach that can provide run-time performance assurance for an online adaptive system.

Based on the literature review and parallel research effort [8], we propose a validation framework for neural network-based online adaptive systems. Figure 3.1 illustrates the framework. As an online adaptive learner, the functional domain of the neural network within the scope of the application can be divided into two distinct parts. In the instance of IFCS, the nominal domain is attended by the Pre-Trained Neural Network (PTNN), which is a Multi-Layer Perceptron Neural Network (MLPNN), referred to as the baseline neural network in Figure 3.1. The off-nominal domain, corresponding to the known failure modes of the aircraft, is the one that causes the online neural network (for example, the DCS neural network in IFCS) to adapt to the domain and accommodate the failures. The adaptation performance causes special concerns and needs validation and verification.

The framework is attempted for validating the learner at both pre-adaptation and post-adaptation phases, as well as its adaptive behavior. Before the learner learns and adapts to environmental changes, such changes and potential failures must be detected, identified

and well-understood before its accommodation. Therefore, at the input layer, we propose to deploy an independent novelty detector to detect novel data/events and potential failure data that might induce violent changes into the learning behavior. During adaptation, we use the stability monitoring tools to continually analyze the online learning process and detect the states that bifurcate away from stable behavior. These monitors provide a visual interpretation of the behavior of learning during online adaptation. After the learner learns on the failure data and adapts to failure conditions, it is used to recuperate against such failure conditions. However, due to the real-time constraint, the learner has to adapt to a new region within a relatively short period of time. Thus, the learning might be inadequate. Consequently, its prediction performance would be poor and unreliable. Therefore, at the output layer, we propose to verify the accommodation performance and evaluate the output data generated by the online adaptive learner by providing a measure of trustworthiness before they propagate into the next component, such as a controller.

Accordingly, as seen in Figure 3.1, we deploy three validation tools, the novelty detector, the online stability monitors, and the validity checker. The novelty detector is designed for detecting potential failures, usually those data/events that are "different enough" from the nominal domain. It provides the system with independent and reliable failure detection results and further validation inferences. As a key step of the validation framework, the runtime stability monitors are developed by Yerramalla et.al. [8, 9]. Its goal is to determine whether, under given flight conditions, the neural network converges, i.e., if its state transition trajectories lead to a stationary state. The online monitor is complemented by mathematical stability proofs [31] that can define its engagement or disengagement. In other words, to preserve computational resources the online monitor may not be engaged in flight conditions that are considered *a priori* safe. After the adaptation, the validity checker is posed for the purpose of evaluating the accommodation performance and verifying the correctness of the output generated by the learner. By each means, necessary notification of possible failure can be conveyed to the controller for caution. We further expect that through our validation tools, rules can be inferred for explaining certain data properties as causes for different learning behavior.

The ultimate goal of our research is that our approach can be extended as a generic model for validating neural network-based online adaptive learning systems.

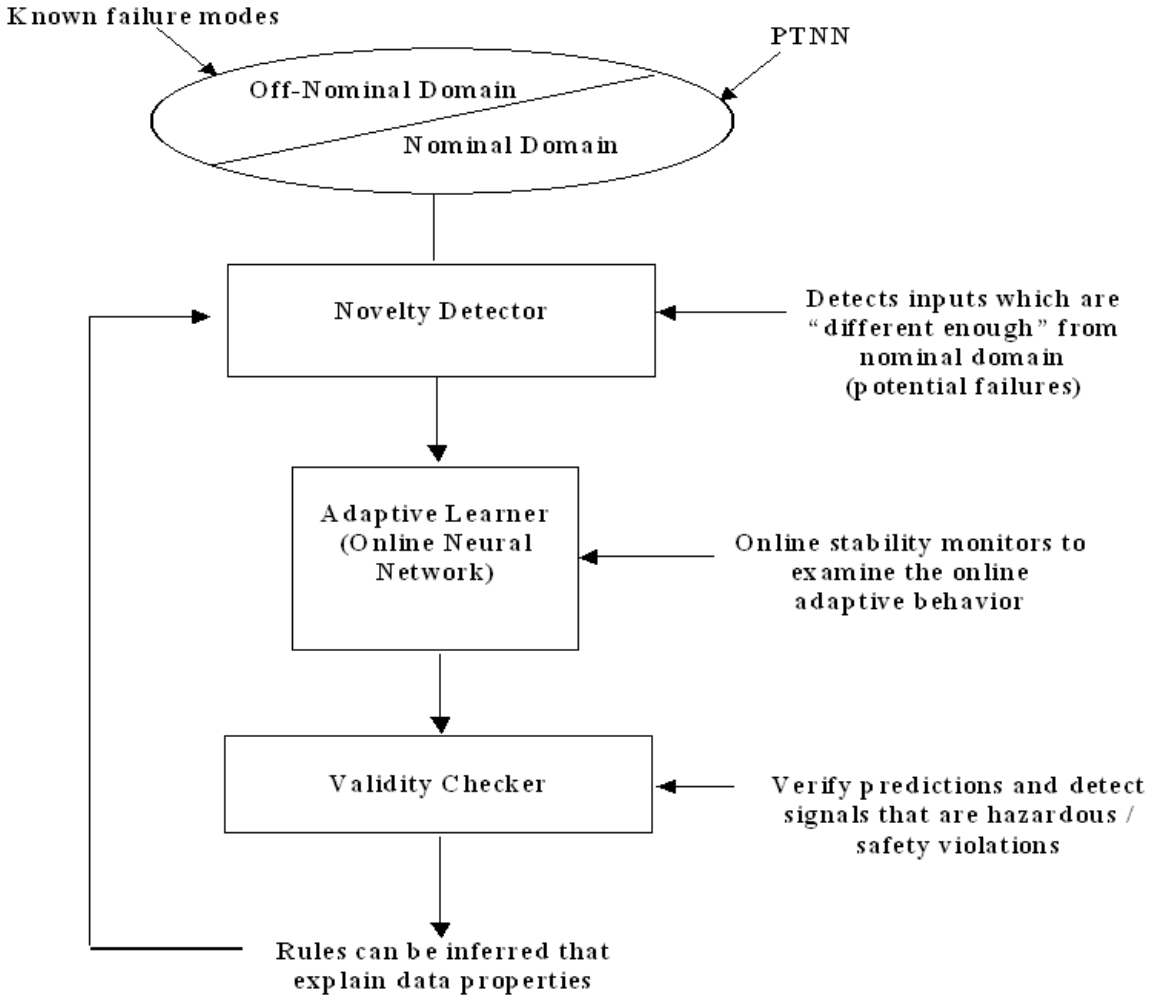


Figure 3.1: A validation framework for online adaptive systems.

Chapter 4

Failure Detection Using Support Vector Data Description

In a safety-critical system like IFCS, a novelty detector can provide failure detection capabilities based on the the duration and the degree of the novel event. We can also use the novelty detector to observe and help understand the impact of the failure conditions on the adaptive learning behavior of the online neural network. An effective and efficient detection method at the input layer offers not only reliable inference on the learning data, but also sufficient response time for manual intervention to take place. For example, when a extremely severe failure occurs, we might have to discard the learning data and prevent it from impairing the performance of the learner. Prompt human operations is required. Efficient evaluation and detection will give the system the freedom to take necessary actions.

Support Vector Data Description (SVDD) has a proven record as a one-class classification tool for novelty detection [60, 61]. In particular, it is highly advantageous in applications whose "abnormal" event/data is extremely costly or near impossible to obtain. We apply SVDD to realize the novelty detection as a major step of our validation approach (See Chapter 3). However, due to space complexity of matrix operations, the optimization process becomes memory and time consuming when n, the size of training set increases. Hence, efficiency needs to be improved for data sets of large size. We present an algorithm that first reduces the space complexity by breaking the training data set into subsets at random and apply SVDD to each subset. Then, based on two lemmas of random sampling and SVDD combining, we merge the data descriptions into a "common decision boundary". Provided with the fact that usually the number of support vectors is relatively few with respect to n, the search for a common decision boundary for a training data set of size n in a $d-$dimension space can be bounded at $O(d^{\frac{3}{2}} n^{\frac{3}{2}} \log n)$ steps.

In this chapter, we first describe the technique and provide a discussion on SVDD. Then in Section 2, we introduce the complexity problem and two important lemmas upon which our fast algorithm is based. The details of our fast SVDD algorithm are presented in Section 3 with proof to its reduced complexity. In Section 4, we present the improved SVDD algorithm and sample experimental results with respect to its feasibility. The results are compared with original solutions. We demonstrate the efficiency as well as the effectiveness of the improved algorithm for novelty detection.

4.1 Support Vector Data Description

Based on Vapnik's Support Vector Machine learning theory [54], Tax et. al. developed support vector data description to solve the one-class classification problem [59, 62]. In order to explain the SVDD, we first give a brief introduction to the support vector machines.

4.1.1 Support Vector Machines

Most real-world classification problems face the difficulty caused by non-separable nonlinear data space. As opposed to designing a complex computational model to approximate the separation, Vapnik derived the support vector machine based on the idea of mapping the data into a higher dimensional space, which is called the feature space, whereby we can define a separating hyperplane [54]. The principal task of the SVM is to find the margin hyperplane that maximizes the minimum distance from the hyperplane to the closest data points (See Figure 4.1).

The search for the optimal separating margin that minimizes the risk of misclassification leads to the use of structural risk minimization . For a given learner, it can be defined by the mapping function $f(x, \alpha)$ where α represents learning parameters. Provided with the probability distribution $P(x, y)$, where (x, y) is the input-target training pair of the l data pairs used for training, the expectation of the test error can be formulated as follows.

$$R(\alpha) = \int \frac{1}{2}|y - f(x, \alpha)|dP(x, y).$$

While $R(\alpha)$ represents the actual risk of a given learner, $R_{emp}(\alpha)$ is defined as the empirical risk that is obtained only from testings on trained data:

$$R_{emp}(\alpha) = \frac{1}{2l}\sum_{i=1}^{l}|y_i - f(x_i, \alpha)|.$$

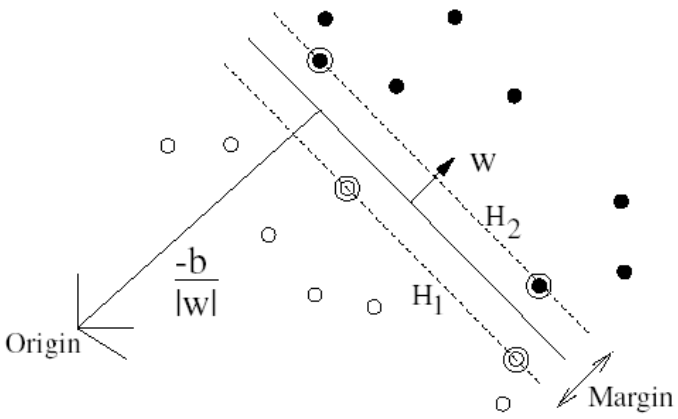


Figure 4.1: The linear separation hyperplanes by SVM for the linear separable. The support vectors are circled [58].

By choosing a certain η, where $0 < \eta < 1$, with probability $1 - \eta$, the following bounds hold [54] :

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{(\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l})}.$$

where h is a non-negative integer that is referred to as the famous VC (Vapnik-Chervonenkis)-Dimension.

Therefore, the generalization ability of this learner depends on the VC dimension of the set of functions $f(x, \alpha)$ that the learning machine implements rather than on the dimensionality of the space. The key concept is that of VC dimension. The VC dimension of a hypothesis space is a measure of the number of different classifications implementable by functions from that hypothesis space.

Two key elements in the implementation of SVM are the techniques of mathematical programming and kernel functions. The parameters are found by solving a quadratic programming problem with linear equality and inequality constraints that can be reduced to a convex optimization problem. The flexibility of kernel functions allows the SVM to search a wide variety of hypothesis spaces. At the same time, there arises the problem of choosing a fitting kernel function, with plenty feasible solutions [67, 58]. The wide spread and successful applications of such a learning machine shows it as a powerful tool for pattern recognition.

4.1.2 Support Vector Data Description

The method of support vector data description originates from the idea of finding a sphere with the minimal volume to contain all data [62]. Given a data set S consisting of N examples $x_i, i = 1, ..N$, the SVDD's task is to minimize an error function containing the volume of this sphere (See Figure 2.2). With the constraint that all data points must be within the sphere, which is defined by its radius R and its center a, the objective function can be translated into the following form by applying Lagrangian multipliers:

$$L(R, a, \alpha_i) = R^2 - \sum_i \alpha_i \{R^2 - (x^2 - 2ax_i + a^2)\},$$

where $\alpha_i > 0$ is the Lagrange multiplier. L is to be minimized with respect to R and a and maximized with respect to α_i. By solving the partial derivatives of L, we also have:

$$\sum_i \alpha_i = 1;$$

and

$$a = \sum_i \alpha_i x_i,$$

which gives the Lagrangian with respect to α_i:

$$L = \sum_i \alpha_i (x_i \cdot x_i) - \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j),$$

where $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. By replacing some kernel functions $K(x, y)$ with the product of (x, y) in the above equations, in particular, the Gaussian kernel function $K(x, y) = exp(- \parallel x - y \parallel^2)/s^2$, we have:

$$L = 1 - \sum_i \alpha_i^2 - \sum_{i \neq j} \alpha_i \alpha_j K(x_i, x_j).$$

By applying kernel functions, we can have a better description of the boundary. The application of kernel functions injects more flexibility to the data description. According to the solution that maximizes L, a large portion of α_i's become zero. Some α_i's are greater than zero and their corresponding objects are those called support objects. Support objects lie on the boundary that forms a sphere that contains the data. Hence, object z is accepted by the description (within the boundary of the sphere) when:

$$\|z - a\|^2 = (z - \sum_i \alpha_i x_i)(z - \sum_i \alpha_i x_i) \leq R^2.$$

Similarly, by applying the kernel function, the formula for checking an object z now becomes:

$$1 - 2\sum_i \alpha_i K(z, x_i) + \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \leq R^2.$$

Since the SVDD is used as a one-class classifier, in practice, there is no actual outliers well defined other than those we randomly draw from the rest of the space outside the target class. Hence, by applying the SVDD, we can only obtain a relatively sound representation of the target class. To detect outliers, more precise criteria should be inferred from empirical testings or pre-defined thresholds. In addition, most real-world data are highly-nonlinear and thus a sphere-like boundary would be almost useless for novelty detection. In order to obtain a "soft boundary", Tax et.al. introduces the parameter C, pre-defined as tradeoff between the volume of our data description and the errors. In general, $C \leq \frac{1}{nf}$, where f is the fraction of outliers that are allowed to fall outside the decision boundary over the total number of data points in S [62]. And the Lagrangian form L is rewritten as:

$$L(R, a, \xi) = R^2 + C\sum_i \xi_i.$$

And the constraints are:

$$\|x_i - a\|^2 \leq R^2 + \xi_i, \forall i.$$

By applying Lagrangian multipliers, the above Lagrangian can be simplified as follows. We are to maximize L with respect to α:

$$L = \sum_i \alpha_i (x_i \cdot x_i) - \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j)$$

with $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i = 1$.

There are three objects obtained from the final solution of maximizing L, which are outliers, support vectors and the rest of the data points that are confined by the boundary. Outliers are those data points that lie outside the boundary. They are considered "novelties" in this case. Support vectors are those data points that sit on the boundary. For support vectors, $0 < \alpha_i \leq C$. A large portion of the data have $\alpha_i = 0$ and these are the data points that lie inside the boundary. Therefore, the center of the hyper-sphere are in fact determined by a very small portion of the data, the support vectors. And the fraction of those data points that are support vectors is a "*leave-one-out estimate of the error on the target data set* [62]." Therefore, we have:

$$E[P(error on the target set)] = \frac{number of support vectors}{N}.$$

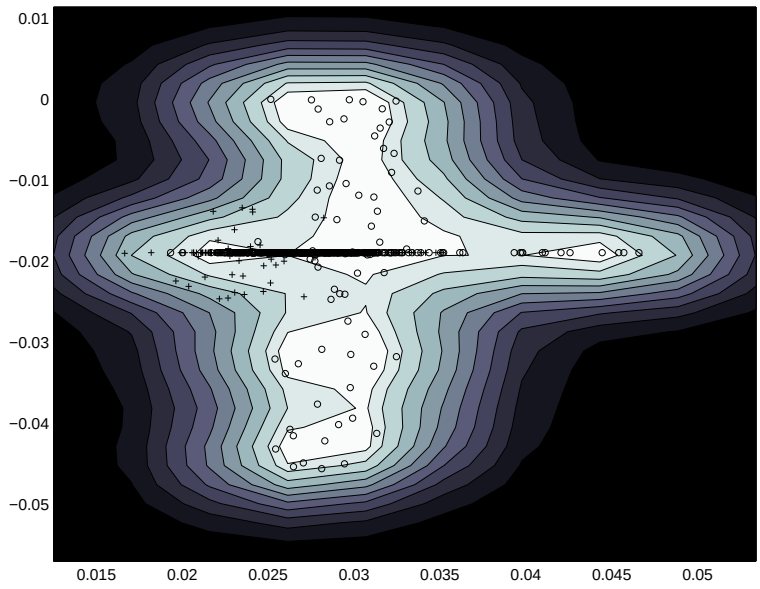


Figure 4.2: Different boundaries of SVDD defined by different thresholds.

Furthermore, SVDD can also produces a "posterior probability-like" novelty measure for each testing data point that falls outside the boundary. Based on the assumption that the outliers are distributed uniformly in the feature space, Tax maps the distance from the outlier object to the defined decision boundary to a novelty measure. It is a quantified measure that indicates the degree of novelty of this particular object with respect to the target class. The mathematical definition of this mapping follows.

$$p(z|O) = exp(-d(z|T)/s)$$

where $p(z|T)$ is the probability that z belongs to the outlier class; $d(z|T)$ is the distance from object z to the decision boundary in the feature space and s is the kernel width.

By applying the SVDD method, we can obtain a sound representation of the target class. To detect outliers (in our case, system failure conditions), a precise criterion should be inferred from empirical testing or pre-defined thresholds. Figure 4.2 illustrates different boundaries of the nominal region. The greater the the distance from the innermost region, the rougher the boundary. Therefore, the sensitivity of outlier detection may be changed. In practice, a pre-defined threshold can be used as the furthest distance of a data point from the center, which the system can tolerate. Such pre-defined thresholds need sufficient testing within each specific data domain.

4.1.3 Advantages vs. Disadvantages

SVDD has been successfully applied to different domains [61, 71, 68]. Based on a close study on the tool, we discuss the advantages and disadvantages with respect to our applications with online adaptive systems. We list the appeals of SVDD as a novelty detection tool as follows.

Advantages:

- As an extension of SVM, SVDD is derived with mathematically rigorous proofs. It is theoretically sound and robust.
- SVDD is free of local minima. As a geometric optimization problem, SVDD provides global solution and avoids the local minimum problem.
- With SVDD, there is few parameters to pre-set. To run a SVDD on a given data set, there are only two parameters need to be pre-defined: s, the kernel width, and C, the fraction of data points that are allowed for misclassification.
- SVDD is less prone to the problem of overfitting than other methods. In a small dimensional space, finding the minimal-volume hyper-sphere containing the normal data points in feature space ensures that the distance between the decision boundaries for distinguishing normal data and novel data is maximized on the side of "normal data" (or class 1). It indicates that the generalization error of the SVDD classifier is bounded. Minimizing the volume reduces the possibility of overfitting.
- The final boundary obtained from SVDD is stable and independent of the algorithm used for the optimization procedure. In other words, using the same data and same parameters, we would obtain the same data description from two different optimization algorithms.
- The detection of novelty can be very simple and straightforward. Since the final shape of the data description only concerns the support vectors, which is a very small fraction of the data points, the evaluation of a testing data point can be extremely simple and fast. SVDD treats the data points inside the boundary equally as "inner points". The α_i's associated with those points are given value 0. Figure 4.3 and Figure 4.4 shows the 2-D demonstration and 3-D contour of the support vector data description on the banana data set respectively. We can see from Figure 4.4 that all data points that lie inside the boundary are on the same contour level.

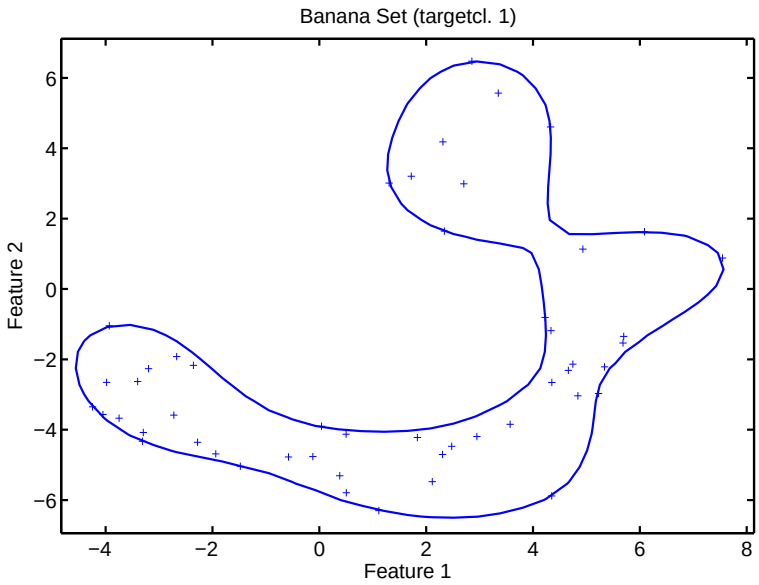


Figure 4.3: Two-dimension demonstration of SVDD on the banana data set.

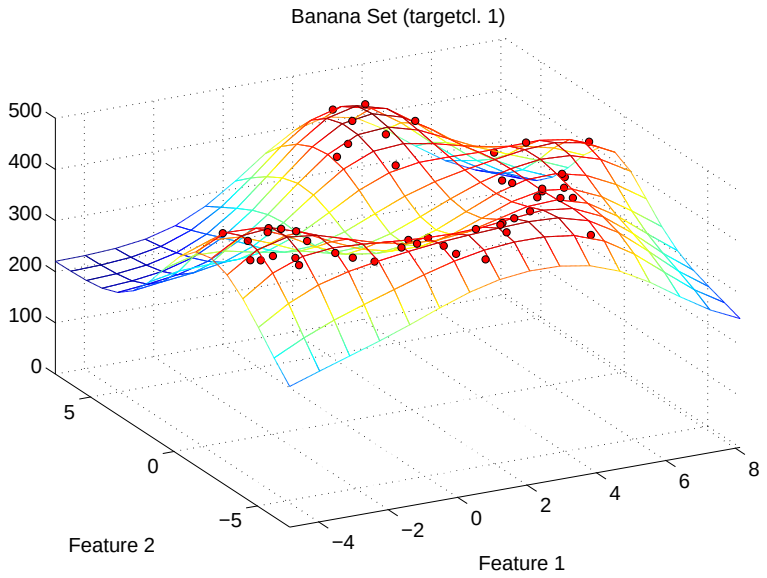


Figure 4.4: Three-dimension demonstration of SVDD on the banana data set.

Although SVDD is considered as a powerful tool for novelty detection with notable advantages as mentioned above, there are several important issues need to be addressed and problems to be solved before they could be successfully applied for novelty detection for an online adaptive system. The drawbacks of SVDD follows.

Disadvantages:

- SVDD still suffers from the curse of dimensionality. Same as SVM, SVDD projects the data points into a feature space using kernel methods. With a large dimensionality, it would not be easy for SVDD to find the structure searching in a large dimensional space. However, if the knowledge about the subspace could be built into the kernel and made available through statistical analysis, the overfitting can be reduced by using adaptive methods to discover such sub-structures and choosing the right kernels.
- The computational cost of SVDD depends on the optimization algorithm it uses. SVDD is a typical geometric optimization problem. Similar to other linear/quadratic programming problems, SVDD also faces the challenge of running time complexity and space complexity in terms of matrix operation. The typical Levengerb-Marquardt optimization algorithm runs at the cost of $O(n^3)$ where n is the size of the data set. In the optimization procedure, when the size of the matrix becomes relatively large, the matrix operations can be very time consuming and thus slow down the algorithm.

The major problem we are facing is the complexity problem associated with SVDD optimization procedure. The implementation of SVDD as online novelty detector becomes inadequate due to algorithm's memory and time limitations. In other words, while SVDD reaches (and surpasses) the desired correctness level in terms of its ability to detect novel data (control failures in IFCS), the algorithm will be computationally too expensive to be run in real-time on a typical on-board computer. A faster SVDD algorithm that can scale to large size data set is needed.

4.2 Solving the Complexity Problem

A simple sampling lemma has been proposed by Gartner et.al. [69] and successfully applied to improve the running time complexity of SVM classifiers [70]. A similar strategy is applied to SVDD algorithm as we break down the optimization procedure into subproblems of smaller size, of which the solution can be found efficiently. Based on the fact that the number of

support vectors is only a very small fraction of the size of the training data set, we derive another lemma, the lemma of combining two different SVDDs, as the basis of the second step of our fast SVDD algorithm. We verified correctness and efficiency of this algorithm. It's time complexity is better than $O(n^3)$, the complexity of the original SVDD learning algorithm. Although we prove that our fast SVDD lowers time and memory complexity, it should be noticed that the combination procedure is performed under the assumption that the number of the support vectors is small and the distribution of the training data is close to normal.

The evaluation step of the SVDD algorithm is highly efficient because it only concerns the support vectors. However, the process of finding the data description boundary can be intimidating in terms of time complexity at $O(n^3)$. It is well-known that the running time complexity due to the optimization procedure for an LP/QP-problem can impede its wide application. In our case, the calculation of the kernel functions became intolerably slow when the size of x reaches a certain number. In a typical run where $n = 800$, the optimization would take approximately 20 minutes on a computer with P4 $1.6GHz$ processor and $256MB$ RAM.

Researchers investigating the complexity problem have tackled the problem from different aspects. In 1997, Osuna et.al. proposed the strategy of decomposition [56] as the first "decomposing" approach to reduce the cost of SVMs. Recently, Balcazar et. al. [70] presented a provably fast training algorithm for SVMs as an interesting application of Gärtner and Welzl's simple random sampling lemma [69]. While most recent research effort has been put on improvement of SVMs, as one of the developers of SVDD, D.M.J. Tax continues his investigation in improving the SVDD algorithm. In his latest work[71], he adopted a incremental learning technique that attempts to minimize the cost of computation at each step by learning one data point at a time while satisfying the Karush-Kuhn-Tucker(KKT) condition in order to preserve optimality, which is practically promising for real-time SVDD. However, the online adaptive system being validated provides data at a frequency of 20Hz and thus we have 200 data points in a 10 second run. Updating the SVDD one data point at a time would not only violate the time constraint but also cause unwanted false positives due to signal oscillations. The desired algorithm should be robust, reliable and compliable with running-time requirements. Inspired by related research contributions, we focus our attention on the random sampling theories and the decomposition strategy.

4.2.1 Two Lemmas

In this section, we present two lemmas as foundations of our fast SVDD algorithm. As important as the following two lemmas, an observation described by Vapnik in [54] states that "*Only support vectors are relevant for the final form of the classifier.*" In other words, without loss of generality, under the assumption of normality, we could obtain a data description of a subset containing all support vectors, that is as same as the data description of the entire data set.

1. **A Simple Random Sampling Lemma.**

   The simple random sampling lemma was derived by Gartner and Welzl and later applied for LP-type problem solving. Among them, fast training algorithms for SVMs were developed and proved by Jose L. Balcazar et.al. in [70]. Below we state the lemma and briefly explain its application for the LP-type problems.

   Let S be a set of size n and ϕ a function that maps any set $R \subseteq S$ to some value $\phi(R)$. We define $V(R) := \{s \in S \setminus R \mid \phi(R \cup \{s\}) \neq \phi(R)\}$ and $X(R) := \{s \in R | \phi(R \setminus \{s\}) \neq \phi(R)\}$. $V(R)$ is the set of violators of R, while $X(R)$ is the set of extreme elements in R. For a random sample R of size r, we denote the expectation of the number of violators and extremes of R by v_r and x_r respectively. The sampling lemma states that:

   Lemma 1. For $0 \leq r \leq n$,

   $$\frac{v_r}{n-r} = \frac{x_{r+1}}{r+1}.$$

   Proof of this lemma is given in [69].

   An abstract notation of LP-type problems is denoted by $L = (S, \phi)$, where S is an finite set of elements and $\phi(R)$ maps $R \subseteq S$ to some value space. With SVDD, the training data set can be regarded as S and $\phi(R)$ maps the subset R to a local bounded region formed by the SVDD, complying with the constraints applied to R.

   For any $R \subseteq S$, a basis of R is a minimal-inclusion subset $Q \subseteq R$ that satisfies $\phi(Q) = \phi(R)$. The combinatorial dimension of L, denoted as $\delta = \delta(L)$, is the size of the largest basis among the bases of all possible $R \subseteq S$. For SVDD, each basis is a

minimal subset of support vectors for some R. Therefore, the combinatorial dimension of an SVDD problem is the largest number of support vectors needed for some R. Note that in normal cases where data is sufficient, the number of support vectors is only a small fraction of the data size [62], we could assume that δ is usually bounded by $\frac{n}{c}$. In [70]'s paper, it can be found that δ is in fact bounded by d, the number of dimensions of the training data.

According to the definition of violators and extremes, for an LP-type problem $L = (S, \phi)$, we have $\phi(R) = \phi(S \setminus V(R))$ and the set of extreme elements of R, $X(R)$ is in fact the intersection of all bases of R. Since $X(R) \leq \delta(L)$, based on the sampling lemma, it has been derived that:

$$v_r \leq \delta \frac{n-r}{r+1}.$$

Furthermore, it has been proven that for a random sample R of size r, when $r \approx \sqrt{\delta n}$, the expected number of violators of R is no more than r [69].

2. **The Combination Lemma.**

Consider now any two random samples $A \subseteq S$ and $B \subseteq S$ for an LP-type problem $L = (S, \phi)$. Let $X(Q)$ and $V(Q)$ denote the set of extreme points and the set of violators of set Q respectively. We define x_Q and v_Q as the expected number of extreme points and violators of set Q. Our combination lemma states:

Lemma 2. Given $X(A)$, $X(B)$ and $V(A)$, $V(B)$, for $C = A \cup B$,

i) $V(C) \subseteq T_1$,
where $T_1 = \{x | x \in V(A) \vee x \in V(B)\}$,
and,

ii) $X(C) \subseteq T_2$,
where $T_2 = \{x | x \in X(A) \wedge x \in X(B)\} \cup \{x | x \in X(A) \wedge x \in V(B)\} \cup \{x | x \in V(A) \wedge x \in X(B)\}$.

We prove the lemma as follows.

Proof. i) $x \in V(C) \Rightarrow x \in V(A \cup B)$
$\Rightarrow x$ is extreme in $\{x\} \cup A \cup B$
$\Rightarrow x$ is extreme in $\{x\} \cup A \vee x$ is extreme in $\{x\} \cup B$

$\Rightarrow x$ violates $A \vee x$ violates B
$\Rightarrow x \in V(A) \vee x \in V(B)$
$\Rightarrow x \in T_1$.

ii) $x \in X(C) \Rightarrow x \in X(A \cup B)$
$\Rightarrow x$ violates $(A \cup B) \backslash x$
Based on proof of **i)**, we induce:
$\Rightarrow x \in V(A \backslash x) \vee x \in V(B \backslash x)$.
There are three cases: $x \in A \wedge x \in B$, $x \notin A \wedge x \in B$, $x \in A \wedge x \notin B$.
$\Rightarrow (x \in X(A) \wedge x \in X(B)) \vee (x \in X(A) \wedge x \in V(B)) \vee (x \in V(A) \wedge x \in X(B))$
$\Rightarrow x \in T_2$.

□

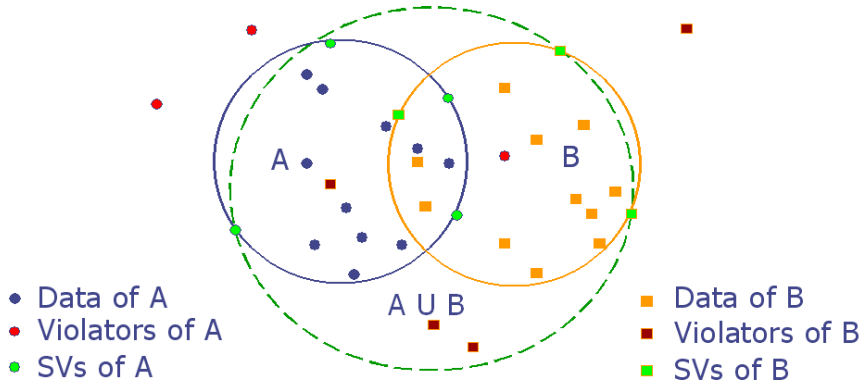


Figure 4.5: An example of combining two SVDDs.

In order to illustrate the application of the above lemmas to an LP-type problem, we consider the problem of smallest enclosing ball, on which the original idea of SVDD was based [59]. Given a set S of n data points in a fixed $d-$dimensional space, we were to find the smallest (in terms of volume) sphere-like shape that can contain all n data points. According to Gärtner et.al. 's proof [69] following the sampling lemma, it is known that for any random sampling set $R \subseteq S$ with r points, the expected number of violators in $S \backslash R$ of R, denoted by p, is no larger than $(d+1)\frac{n-r}{r+1}$. In the case of $r = n/2$, we obtain $p < d+1$. While $r \approx \sqrt{dn}$, it can be easily proven that $p < r$.

Figure 4.5 is a simple example of combining two different sphere shape data descriptions in a two dimensional space. The dotted line represents the smallest enclosing ball for

the data set $A \cup B$. The subset relations between $V(A \cup B)$ and $V(A) \cup V(B)$ can be found in the plot, as well as that of $X(A \cup B)$ and T_2. Obviously, we can restrict our T_1 to $(V(A) \cup V(B)) \cap \overline{A \cup B}$.

4.2.2 The Fast SVDD Training Algorithm

Based on the fact that only support vectors concern the final form of the classification, we modify the original algorithm in spirit of "decomposing and combining". In the step of decomposing, the simple random sampling lemma can then be applied in order to enforce the bound on the expected number of extreme points and the expected number of violators. A strategy similar to "divide and conquer" is suitable for pairwise merging of different data descriptions. This procedure of "combining" can be bounded at an expected $O(\log n)$ steps according to our Lemma 2.

Figure 4.6 describes our fast SVDD algorithm. Note that in this algorithm solutions to subproblems are still obtained through the original SVDD algorithm.

We notice that the improvement of the algorithm relies on the fact that by choosing $m \approx \sqrt{dn}$, the number of violators is bounded at m and the algorithm will converge to the global solution in a finite number of iterations. In the worst case scenario where the entire data set has to be learned, the number of iterations of our algorithm is $\sqrt{\frac{n}{d}}$, and for each step of decomposition and combination, the common decision boundary is found at a cost of $O(d^{\frac{3}{2}} n^{\frac{3}{2}})$. Therefore, the expected running complexity in total is $O(dn^2)$. However, we can expect the iteration to be ended within $\log n$ steps and thus we can reach a running time of $O(d^{\frac{3}{2}} n^{\frac{3}{2}} \log n)$. On the other hand, since the size of the problem can be shrunk down to a level that computers can handle it at a reasonable speed, the memory utilization becomes efficient and less error-prone. Yet, m is not strictly fixed at $\sqrt{dn}$. Instead, it can be any number that is larger than the expected number of support vectors but "optimally small" in terms of system specifications. The convergence will be reached in a finite number of steps due to the fact the objective function is convex and quadratic. However, the analysis of complexity in these cases might be slightly different.

4.3 Experiments and Discussion

In order to test our fast SVDD algorithm, we conduct two experiments. One uses two data sets whose distributions are approximately the same. The other uses two data sets that come from two distinct distributions.

1. $W_1 \leftarrow$ **arbitrarily select** m **points from** S**;**

Find the solution to W_1**, denote it by** P_1**;**

$X_1 \leftarrow$ **the support vectors of** (W_1, P_1)**,** $x_1 \leftarrow |X_1|$**;**

$W_2 \leftarrow$ **select** m **points from** $S \setminus W_1$**;**

2. Find the solution to W_2**, denote it by** P_2 **;**

$X_2 \leftarrow$ **the support vectors of** (W_2, P_2)**,** $x_2 \leftarrow |X_2|$**;**

3. $V_1 \leftarrow$ **the violators of** (W_1, P_1)**,** $v_1 \leftarrow |V_1|$**;**

if $v_1 = 0$**, return** (S, P_1) **as the final solution.**

$V_2 \leftarrow$ **the violators of** (W_2, P_2)**,** $v_2 \leftarrow |V_2|$**;**

if $v_2 = 0$**, return** (S, P_2) **as the final solution.**

4. $R \leftarrow (X_1 \cup X_2) \cup ((V_1 \cup V_2) \cap \overline{W_1 \cup W_2})$**;**

if $|R| < m$**, add** $m - |R|$ **points from** $W_1 \cup W_2$ **into** R**;**

find the solution to R**, denoted by** Q**,**

$X \leftarrow$ **support vectors of** (R, Q)**;**

5. $V \leftarrow$ **the violators of** (R, Q) **in** $S \backslash (W_1 \cup W_2)$**;**

if $|V| = 0$

return (S, Q) **as the final solution.**

if $|V| > m$

$W_2 \leftarrow$ **randomly sample** m **points from** V**;**

else

$W_2 \leftarrow V \cup \{$ **randomly sampled** $m - |V|$ **points from** $S \backslash V$ $\}$**;**

end

6. $W_1 \leftarrow X \cup \{$ **randomly sampled** $m - |X|$ **points from all learned non-SV, non-violator points** $\}$**;**

7. Repeat $2 - 6$ **until global solution is found.**

Figure 4.6: A decomposing and combining algorithm for SVDD.

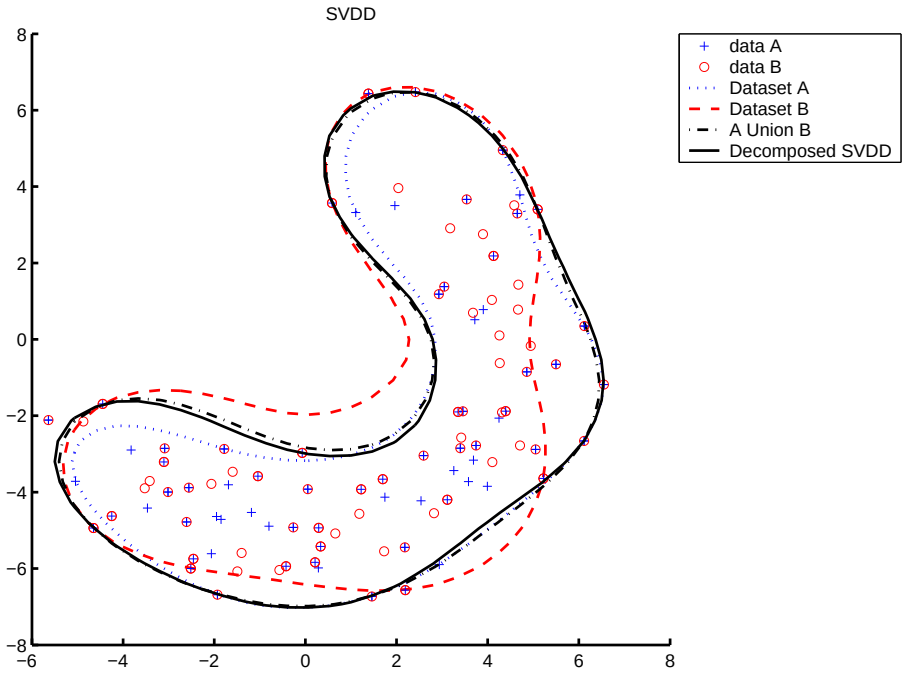


Figure 4.7: Compare fast SVDD and original SVDD on two data sets. These two data sets come from the same distribution.

The results obtained from the first experiment are illustrated in Figure 4.7. We generated two banana data sets, data set A and data set B. Each has 50 data points. There are four different runs in our experiment: 1) We first run the original SVDD algorithm on data set A; 2) Then we run the SVDD on data set B; 3) we then mix these two data sets and run the SVDD algorithm on the union of A and B; 4) At last, we run our fast SVDD algorithm on A and B to obtain another data description. The results are presented in Figure 4.7. In the last step of combining, those data points including the extreme points and violators and those points that are randomly chosen from A and B are represented by the circles with crosses embedded. Figure 4.8 compares the time recorded for each run. Figure 4.7 shows four different data descriptions obtained from four different runs. Notice that the boundary obtained from 3) and the boundary obtained from 4) are almost identical. However, the time consumed by 4) is as less as half of the time needed for complete 3).

It also can be seen in Figure 4.7 that all the extreme points and violators of data set A and data set B are indeed included in the "combining" step and contribute to the final solution of the data description of $(A \bigcup B)$.

Figure 4.9 illustrates the results for the second experiment. In this experiment, we still

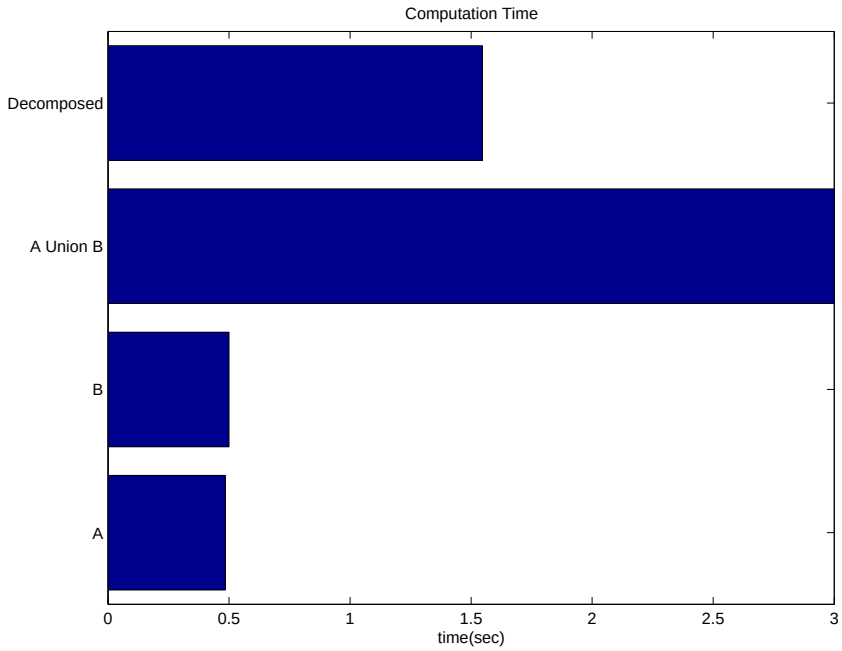


Figure 4.8: Time recorded for the experiment.

use the banana data as data set A. However, we choose a gaussian distributed data set as data set B. Each data set contains 50 data points. We run the same procedure as in experiment 1 and obtain four different boundaries as shown in Figure 4.9. This time, we notice the data description obtained from our fast SVDD is much tighter than that obtained from the original SVDD on $(A \bigcup B)$. This is caused by the fact that different width are needed for these two data sets to obtain their fair data descriptions. Our fast SVDD algorithm chooses the smaller width setting while the original SVDD selects the larger width setting and thus obtain a rougher boundary.

We further test the efficiency of our fast SVDD algorithm by running experiments on data sets with different sizes. Table 4.1 lists the time recorded for our experiments. In Table 4.1, rows show the execution time in seconds for each run of two algorithms on different size data sets. As n increases, the difference between computation times of these two algorithms increases. When n reaches 1000, our fast SVDD algorithm becomes more than 100 times faster than the original one.

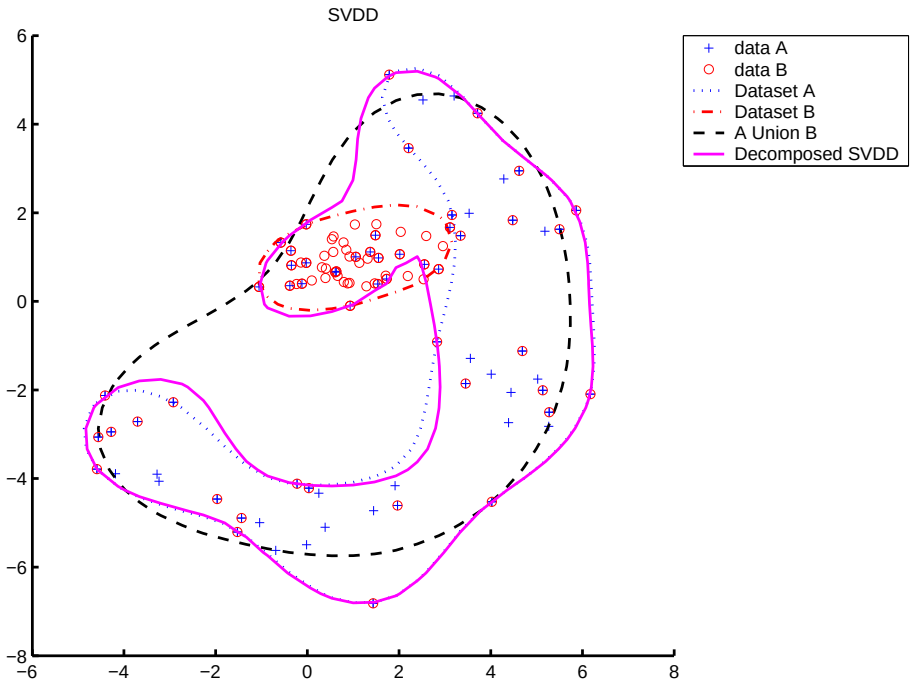


Figure 4.9: Compare fast SVDD and original SVDD on two data sets. These two data sets come from two different distributions.

Table 4.1: Running time recorded for two algorithms

Data size (n)	Fast SVDD	SVDD
200	4	240
400	12	1257
800	25	2471
1000	36	3645

4.3.1 Discussion

We have shown the improved efficiency of our fast SVDD algorithm. However, there are several issues we must deal with while using the fast SVDD algorithm. We discuss them as follows.

- Recall that in the SVDD algorithm, the parameter C is introduced as the trade-off between the volume of the hypersphere and the error. It is set as the fraction of data points that are allowed for misclassification. For example, if we have 1000 data points in S, the fraction C is set to be 10%. Thus, the number of points that are allowed to be outside the boundary is 100. After we break down S into ten small working sets and run SVDD on each set, we combine the solutions. However, during the process of combining, it is hard to obtain the same solution while maintaining the C for the entire data set. It is because whenever we combine, we continuously deal with a data set of size 100. Hence, the parameter C needs to be justified for the combining process. However, there is no ultimate answer to *"how to find an optimal solution to change C"* but through empirical search.

- As illustrated in Figure 4.9, it is very likely that our fast SVDD will offer a tighter description than that by the original algorithm. Although this might not be a problem as we adopted the random sampling scheme, it still causes trouble for cases where sequential combinations are needed. For instance, when online data description is needed, we won't be able to apply the random sampling to the data sets. Therefore, the chances are the incoming data set might have a very different distribution. Our results suggest that proper setting of width must be determined for such a situation. Further, for certain situations where a tighter boundary is needed, fast SVDD can still be a better choice.

4.4 Summary

We investigate the support vector data description method and discuss its advantages as well as problems. SVDD has many appealing features as a very promising novelty detection tool. However, SVDD faces the challenge of space and time complexity when the data size reaches a certain number and the matrix operation becomes extremely time-consuming. In reality, the control system of IFCS runs at the speed of $20Hz$ and generates 200 data points within 10 seconds. A large amount of data is needed to obtain a sound and meaningful data description for the nominal regions. Hence, we must improve the efficiency of SVDD to be used as a potential tool for novelty detection to realize a major step of our validation approach.

Based on two lemmas, the random sampling lemma and the combining lemma, we develop a fast SVDD algorithm. The improved SVDD algorithm consists of two major steps, i.e., decomposition and combination. As an interesting application of the simple random sampling lemma and the combination lemma, our fast SVDD algorithm demonstrates improvement of efficiency with respect to run-time complexity and memory utilization. In the context of previously proposed V&V approach for online adaptive systems, the fast SVDD algorithm can be applied to defining boundaries for "nominal" regions and used for real-time failure detections.

Chapter 5

Validity Index in Dynamic Cell Structures

The validation at the output layer of an online adaptive system is as equally important as the novelty detection at the input layer. The validity checker verifies the adaptation performance at the output layer and prevents the unreliable output from entering the next component, which usually is the controller/actuator of the system. It measures the "trustworthiness" of each output and alerts operators when an output is considered potentially hazardous in terms of "reliability and trustworthiness".

Often viewed as black box tools, neural network models have been successfully applied in various fields. In safety-critical systems such as flight control, neural networks are adopted as a popular soft-computing paradigm to carry out the adaptive learning. The appeal of including neural networks in these systems is in their ability to cope with a changing environment. Unfortunately, the validation of neural networks is particularly challenging due to their complexity and nonlinearity and thus reliable prediction performance of such models is hard to assure. The uncertainties (low confidence levels) existed in the neural network predictions need to be well analyzed and measured during system operation. In essence, a reliable neural network model should provide not only predictions, but also confidence measures of its predictions.

As a special type of Self-Organizing Maps, the Dynamic Cell Structures (DCS) network has topology-preserving adaptive learning capabilities that can, in theory, respond and learn to abstract from a much wider variety of complex data manifolds. However, the highly complex learning algorithm and non-linearity behind the dynamic learning pattern pose serious challenge to validating the prediction performance of DCS and impede its spread in control applications, safety-critical systems in particular.

In this chapter, we propose a mechanism to generate the measures of "trustworthiness" for DCS predictions. We present the validity index, an estimated confidence interval associated with each DCS output, as a reliability-like measure of the network's prediction performance. Our experiments demonstrate that the validity index in DCS is a feasible validation approach.

5.1 The Dynamic Cell Structures

The Dynamic Cell Structures (DCS) network is derived as a dynamically growing structure in order to achieve better adaptability. DCS is proven to have topology-preserving adaptive learning capabilities that can respond and learn to abstract from a much wider variety of complex data manifolds [72, 73]. The structural flexibility of DCS network has gained it a good reputation of adapting faster and better to a new region. A typical application of DCS is the NASA Intelligent Flight Control System (IFCS). DCS is employed in IFCS as online adaptive learners and provides derivative corrections as control adjustments during system operation, and it has been proven to outperform Radial Basis Function (RBF) and Multi-Layer Perceptron network models [29, 74]. As a crucial component of a safety critical system, DCS network is expected to give robust and reliable prediction performance in operational domains.

The Dynamic Cell Structure (DCS) network can be seen as a special case of Self-Organizing Map (SOM) structures. The SOM is introduced by Kohonen [75] and further improved to offer topology-preserving adaptive learning capabilities that can, in theory, respond and learn to abstract from a much wider variety of complex data-manifolds. The DCS network adopts the self-organizing structure and dynamically evolves with respect to the learning data. It approximates the function that maps the input space. At last, the input space is divided into different regions, referred to as the Voronoi regions [72, 73, 76]. Each Voronoi region is represented by its centroid, a neuron associated with its reference vector known as the "best matching unit (bmu)". Further, a "second best matching unit (sbu)" is defined as the neuron whose reference vector is the second closest to a particular input. Euclidean distance metric is adopted for finding both units. The set of neurons connected to the bmu are considered its neighbors and denoted by nbr.

The training algorithm of the DCS network combines the competitive Hebbian learning rule and the Kohonen learning rule. The competitive Hebbian learning rule is used to adjust the connection strength between two neurons. It induces a Delaunay Triangulation into the network by preserving the neighborhood structure of the feature manifold. Denoted by

$C_{ij}(t)$, the connection between neuron i and neuron j at time t is updated as follows:

$$C_{ij}(t+1) = \begin{cases} 1 & (i = bmu) \wedge (j = sbu) \\ 0 & (i = bmu) \wedge (C_{ij} < \theta) \\ & \wedge (j \in nbr \setminus \{sbu\}) \\ \alpha C_{ij}(t) & (i = bmu) \wedge (C_{ij} \geq \theta) \\ & \wedge (j \in nbr \setminus \{sbu\}) \\ C_{ij}(t) & (i, j \neq bmu) \end{cases}$$

where α is a predefined forgetting constant and θ is a threshold preset for dropping connections.

The Kohonen learning rule is used to adjust the weight representations of the neurons which are activated based on the best-matching methods during the learning. Over every training cycle, let $\Delta\vec{w}_i = \vec{w}_i(t+1) - \vec{w}_i(t)$ represent the adjustment of the reference vector needed for neuron i, the Kohonen learning rule followed in DCS computes $\Delta\vec{w}_i$ as follows.

$$\Delta\vec{w}_i = \begin{cases} \varepsilon_{bmu}(\vec{m} - \vec{w}_i(t)) & (i = bmu) \\ \varepsilon_{nbr}(\vec{m} - \vec{w}_i(t)) & (i \in nbr) \\ 0 & (i \neq bmu) \wedge (i \notin nbr) \end{cases}$$

where $\vec{m}$ is the desired output, and $0 < \varepsilon_{bmu}, \varepsilon_{nbr} < 1$ are predefined constants known as the learning rates that define the momentum of the update process. For every particular input, the DCS learning algorithm applies the competitive Hebbian rule before any other adjustment to ensure that the sbu is a member of nbr for further structural updates.

The DCS learning algorithm is briefly described in Figure 5.1. According to the algorithm, N is the number of training examples. Resource values are computed at each epoch as local error measurements associated with each neuron. They are used to determine the sum of squared error of the whole network. Starting initially from two connected neurons randomly selected from the training set, the DCS learning continues adjusting its topologically representative structure until the stopping criterion is met. The adaptation of lateral connections and weights of neurons are updated by the aforementioned Hebbian learning rule and Kohonen learning rule respectively. The resource values of the neurons are updated using the quantization vector. In the final step of an iteration, the local error is reduced by inserting new neuron(s) in certain area(s) of the input space where the errors are large. The whole neural network is constructed in a dynamic way such that in the end of each learning epoch, the insertion or pruning of a neuron is triggered when necessary.

A DCS network has two operational modes.

Initialization;

Repeat until stopping criterion is satisfied;

{

Repeat N **times**

{

Determine the bmu and sbu;

Update lateral connections;

Adjust the weights;

Update resource values;

}

If needed, a new neuron is inserted;

Decrement resource values;

}

Figure 5.1: A brief description of the DCS learning algorithm.

- DCS in learning. The DCS network fetches data from the data buffer and keeps learning using the algorithm in 5.1. During learning, the DCS network adjusts its structure to achieve a sound mapping of the current operational domain.

- DCS in recall. In the recall mode, a DCS network is used for prediction. After adaptation, as a mapping function, the DCS network is then employed to recall parameter values at any chosen dimension. It should be noted that the computation of an output is different from that during training. When DCS is in recall, the output is computed based on two neurons for a particular input. One is the bmu of the input; the other is the closest neighbor of the bmu other than the sbu of the input. In the absence of neighboring neurons of the bmu, the output value is calculated using the bmu only.

5.2 The Validity Index in DCS networks

As a V&V method, validity check is usually performed through the aide of software tools or manually to to verify the correctness of system functionality and the conformance of system performance to pre-determined standards. The validity index proposed by J. Leonard [33] is a reliability-like measure provided for further validity checking. Validity index is a confidence interval associated with each output predicted by the neural network. Since a poorly fitted region will result in lower accuracy, it should be reflected by poor validity index and later captured through validity checking.

Our research effort is dedicated to validating and improving the prediction performance of DCS network by investigating the confidence for DCS outputs. We present the Validity Index, as a measure of accuracy imposed on each DCS prediction. Each validity index reflects the confidence level on that particular output. The proposed method is inspired by J. Leonard's paper in validity index for validating Radial Basis Function (RBF) neural networks [33]. He generates a reliability-like measure as the validity index for each output based on statistical analysis. Different from the pre-defined static RBF network structure, the DCS progressively adjusts (grows/prunes) its structure including locations of neurons and connections between them to adapt to the current learning data. Thus, unbiased estimation of confidence interval is impossible to obtain through S-fold cross-validation due to constraints of time and space. Yet, DCS emphasizes topological representation of the data, while RBF does not. By the end of DCS learning, the data domain is divided into Voronoi regions. Every region has a neuron as its centroid. The "locality" of DCS learning is such that the output is determined by only two particular neurons, the best matching unit

and the second best matching unit. Intuitively, if the Voronoi region of a neuron does not contain sufficient data, it is expected that the accuracy in that region will be poor. Based on the "local error" computed for each neuron, our approach provides an estimated confidence interval, called the Validity Index for DCS outputs.

The validity index in DCS networks is defined as an estimated confidence interval with respect to a DCS output given a testing input. It can be used to model the accuracy of the DCS network fitting. The computation of a validity index for a given input x consists of two steps:

1. compute the local error associated with each neuron, and
2. estimate the standard error of the DCS output for x using information obtained from step 1).

The detailed description of these two steps follows.

Figure 5.2 illustrates the final form of DCS network structure, represented by neurons as centroids of Voronoi regions. Since the selection of the best matching unit must be unique, only those data points whose bmu are the same will be contained in the same region. Therefore, all Voronoi regions are non-overlapping and cover the entire learned domain. The data points inside each region significantly affect the local fitting accuracy. The local estimate of variance of the network residual in a particular region can be calculated over these data points contained in the region and then be associated with its representative neuron. More specifically, the local estimate of variance s_i^2 associated with neuron i can be computed as:

$$s_i^2 = \frac{1}{(n_i - 1)} \sum_{k=1}^{n_i} E_k,$$

where n_i is the number of data points covered by neuron i and E_k is the residual returned from the DCS recall function for data point k.

In Section 1, we showed that the adjustment by competitive Hebbian learning rule concerns connections only between the bmu and its neighbors. The further update of weight values by Kohonen learning rule is performed only on the bmu and its neighbors. Consequently, training data points covered by the neighboring neurons of neuron i make proportional contributions to the local error of neuron i. Considering such contributions, we modify the computation of the local estimate of variance, now denoted $s_i^{'2}$, as follows.

$$s_i^{'2} = \frac{s_i^2 + \sum_{j \in nbr} C_{ij} s_j^2}{1 + \sum_{j \in nbr} C_{ij}}.$$

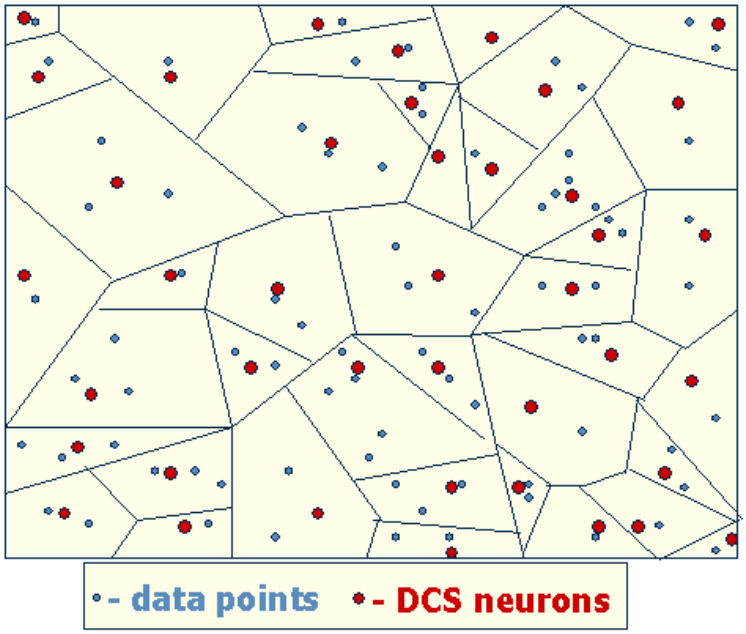


Figure 5.2: Voronoi regions of a DCS network.

As a result, the influence of all related data points is taken into account accordingly based on connections, referred to as C_{ij}, between the bmu and its neighbors. It should be noted that since the DCS networks within IFCS are trained online, no cross-validation is allowed. Hence, the residual calculated for each data point is in fact a biased estimate of the expected value of residual due to the fact that each data point itself contributed to its own prediction. Nonetheless, under the assumption that there is no severe multicollinearity and relatively few outliers exist in the data, the probability that the deviation from the expected value will be significant is very low and thus can be ignored.

Recall that the output produced by DCS is determined by the bmu and its closest neighbor (cnb) of the given input. Thus, the local errors associated with these two neurons are the source of inaccuracies of fitting. We use the standard error, a statistic that is often used to place a confidence interval for an estimated statistical value. Provided with the the local estimate of variance for every neuron from step 1), we now define the 95% confidence limit for the local prediction error estimate with respect to neuron i as:

$$CL_i = t_{.95}\sqrt{1 + \frac{1}{n_i}}s'_i,$$

where $t_{.95}$ is the critical value of the Student's t-distribution with $n_i - 1$ degrees of freedom.

The 95% confidence interval for the network output y given a testing input is thus given by:

$$(y - \frac{(CL_i + CL_j)}{2}, y + \frac{(CL_i + CL_j)}{2}),$$

where $i = bmu$ and $j = cnb$ with respect to input x.

Now we slightly modify the DCS training algorithm in order to calculate the validity index. Note that because all needed information is already saved at the final step of each training cycle, without any additional cost we simply calculate $s_i'^2$ for each neuron after the learning stops. When the DCS is in recall, the validity index is computed based on the local errors and then associated with every DCS output. In order to complete the validity check, further examination needs to be done by software tools or system operators. In the case of IFCS, a domain specific threshold can be pre-defined to help verify that the accuracy indicated by the validity index is acceptable according to certain standards. And this check can be either automated or completed manually by the operator.

5.3 An Example with Artificial Data

In order to demonstrate the validity index in DCS network model as an improvement of the network prediction, we present an example using an artificial data set. The DCS is trained on a single-input, single-output function as seen in [33]:

$$f(x) = 0.2\sin(1.5\pi x + 0.5\pi) + 2.0 + \varepsilon,$$

where ε is a Gaussian noise.

We sample x's from the interval $[-1, 1]$ randomly therefore there would be some regions where the learning data points are not as dense as those in others. We then obtain two different DCS network models by varying the stopping criterion. Figure 5.3 and Figure 5.4 illustrate the validity index for these two DCS models, one with 13 neurons and the other with 27 neurons, respectively. By comparing the prediction performance of these two models using the validity index, which is shown as confidence band in both figures, we can conclude that the DCS network model shown in Figure 5.4 has better prediction performance. Furthermore, we can observe that regions with sparse learning data have low confidence levels.

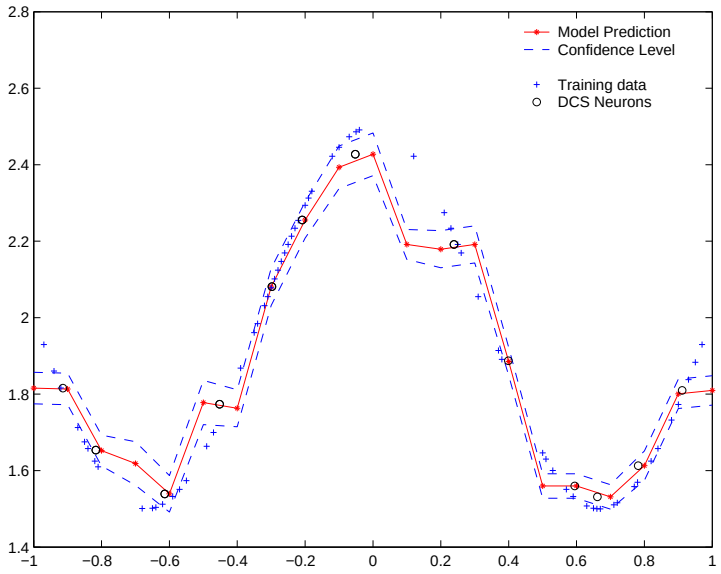


Figure 5.3: Example of validity index for a DCS model with 13 neurons.

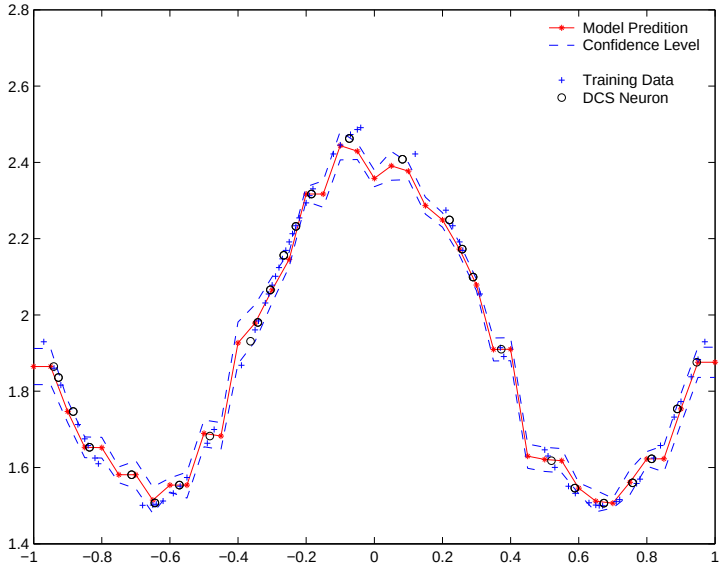
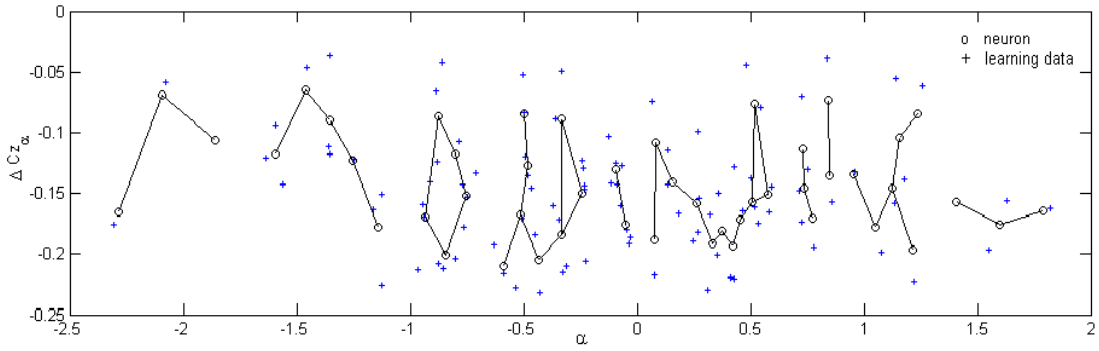
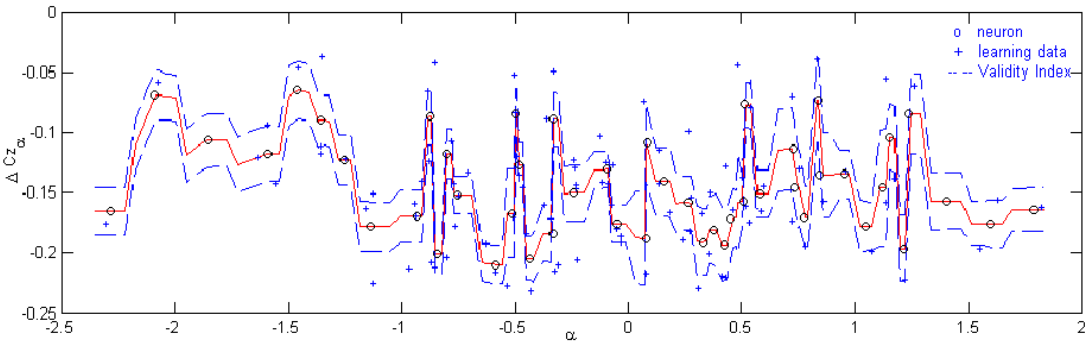


Figure 5.4: Example of validity index for a DCS model with 27 neurons.



(a)



(b)

Figure 5.5: Validity Index in DCS with 48 neurons. (a): Final form of the DCS network after training. (b): DCS outputs and associated VI.

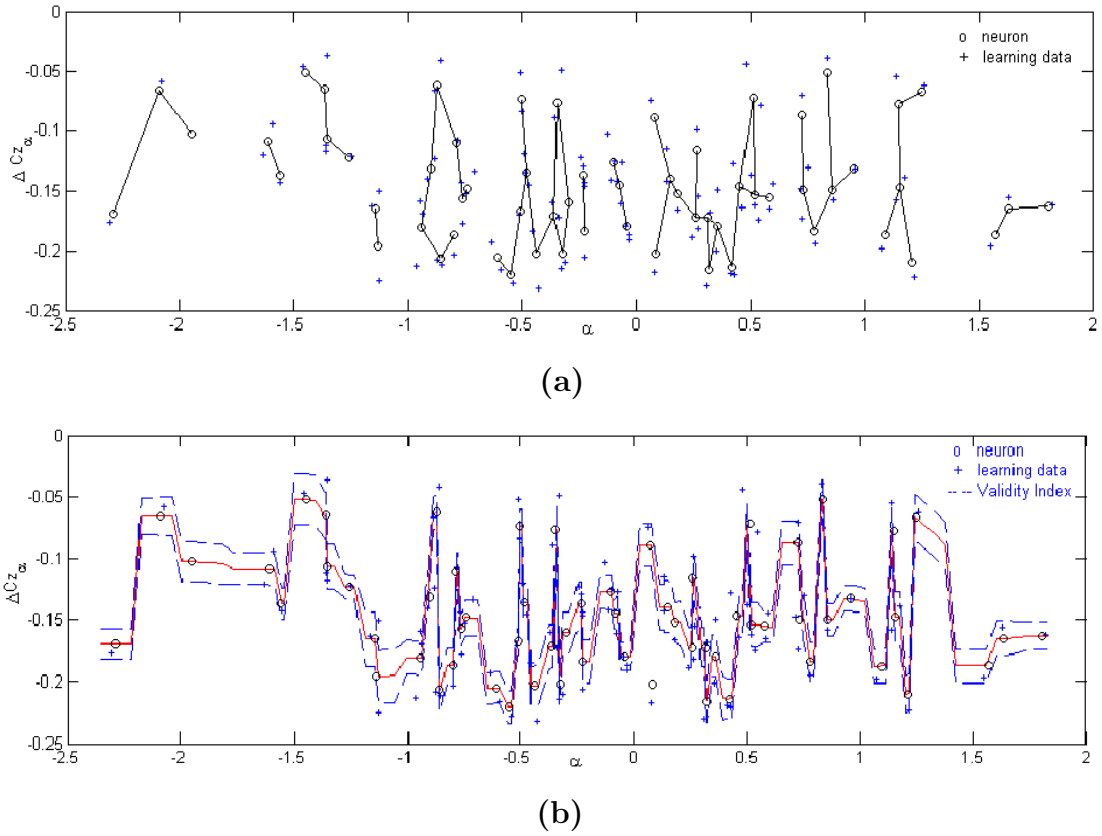


Figure 5.6: Validity Index in DCS with 62 neurons. (a): Final form of the DCS network after training. (b): DCS outputs and associated VI.

5.4 A Case Study on IFCS

The experiments we conducted use real-valued data collected from an experimental F-15 flight control simulator for IFCS. The control framework of the simulator is based on the IFCS architecture, shown in Figure 1.1.

The simulation data used for illustrating the validity index is a segment of nominal flight condition data. It contains approximately five seconds of flying time corresponding to a typical 100 frames of data at the simulation rate of 20Hz. A data frame is a point in a six-dimensional space corresponding to four sensor readings (independent variables) and two stability and control derivative errors from PID and PTNN (dependant variables). The DCS network tested here is the $DCS - C_z$ network, one of the five DCS-subnetworks of the IFCS. The independent variables are Mach number (the ratio of the speed of the aircraft to the local speed of sound), α (aircraft's angle of attack), β (aircraft's pitch rate), and altitude of the aircraft. The dependent variable are ΔCz_α and ΔCz_β, two stability and control derivative errors generated by the difference between PID and PTNN.

In this experiment, we select α and ΔCz_α as two dimensions of our training and testing data. First, the DCS network is trained on 100 training data points. Two DCS networks were trained over the same data set yet stopped at different thresholds of model residuals. One contains 48 neurons and the other has 62 neurons. Their structures are illustrated by Figure 5.5(a) and Figure 5.6(a), respectively. For all four plots in Figure 5.5 and Figure 5.6, training data points are represented by crosses in both plots, with x-axis representing α and y-axis representing ΔCz_α. DCS neurons are represented by dark circles connected with their neighbors by solid lines. Each network was then tested over uniformly randomly selected data points over the learned input space. During testing, the ΔCz_α was calculated as output. And the validity index is shown as confidence band for the DCS output in both Figure 5.5(b) and Figure 5.6(b).

What can be seen from both figures is that a much wider confidence band can be found in regions that lack data. It shows the validity index is able to reflect the increasing uncertainties of the network output due to insufficient training data. In the case of IFCS, it is important to capture such uncertainties. Without checking its validity, the use of unreliable outputs could result in the controller delivering misleading control dynamics that might cause failures. Furthermore, while comparing the validity index of two networks (See Figure 5.5(b) and Figure 5.6(b)), we notice that when the number of neurons increases, the validity index of network outputs improves, which is theoretically true. By adding more neurons into regions having larger residual values, the overall model residual is reduced and the local

error associated with each neuron is thus reduced correspondingly.

5.5 Summary

Known for its structural flexibility, DCS networks are adopted in safety-critical systems for online learning in order to quickly adapt to a changing environment and provide reliable outputs when needed. However, DCS network predictions cannot be constantly trusted because locally poor fitting will unavoidably occur due to extrapolation.

As the final step of our validation approach, we propose the validity index in DCS to examine the trustworthiness for DCS output. The implementation of validity index is straightforward and does not require any additional learning. The experimental results obtained from artificial data and IFCS simulation data demonstrate the effectiveness of such a method. The computed validity index effectively indicates poor fitting within regions characterized by sparse data. We are confident that the validity index in DCS is a feasible approach for realizing the validity check at the output layer of IFCS.

Chapter 6

Experimental Results

We evaluate the proposed validation methods through experiments with the Intelligent Flight Control System (IFCS), introduced in Chapter 1. The IFCS is an adaptive flight control application for NASA F-15 aircraft. Through a high-fidelity IFCS simulator [77], we were able to collect not only nominal flight data but also simulated failure data to test and evaluate our techniques.

6.1 Data Collection

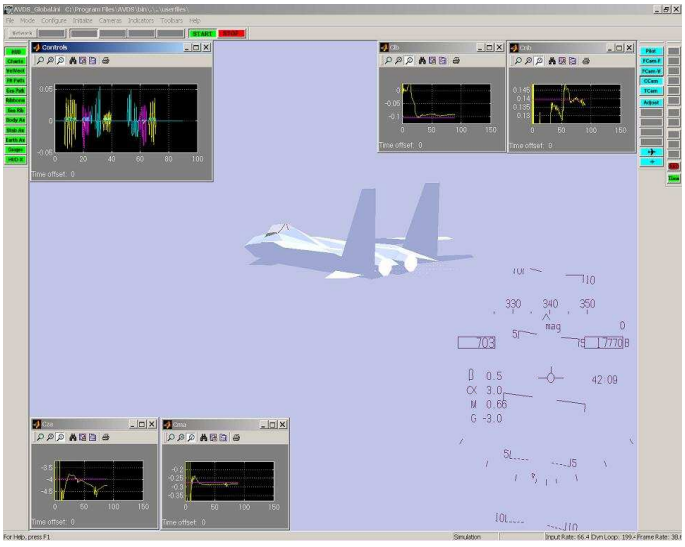


Figure 6.1: NASA-WVU F-15 Simulator

Major advances in development of modern control laws have generated the need for developing very detailed and sophisticated simulation environments for research and development purposes. A simulation tool has been developed at West Virginia University [77] for on-line aircraft parameter identification with a specific fault tolerant flight control scheme used in the NASA IFCS F-15 program. The simulation package is modular and flexible so that different methods can be used for each of the tasks of the general fault tolerant control system, such as aircraft model, controller, parameter identification method, and on-line data storage. The user has various simulation options through specific GUIs and can select different control loop configurations, parameter identification methods, and failure scenarios.

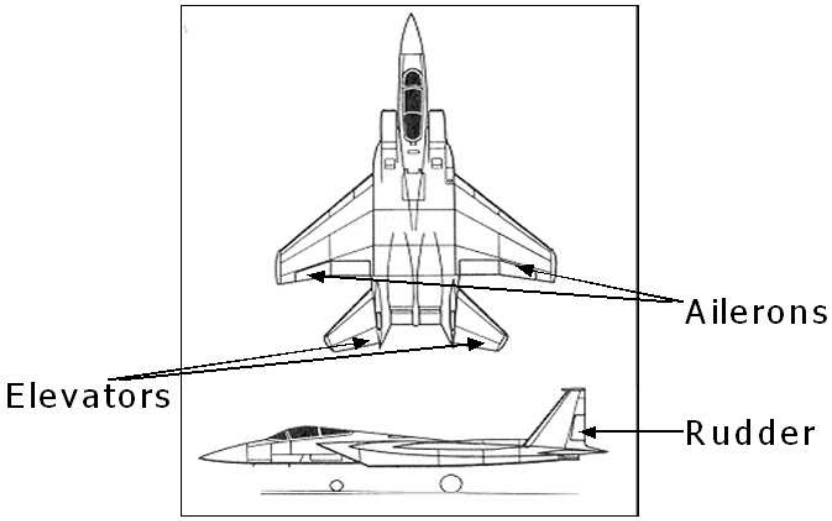


Figure 6.2: Primary control surfaces on F-15 aircraft.

With the aide of the high-fidelity flight control simulator, we are able to test our approach for adaptive flight control through experimentation in simulated environments. Figure 6.1 shows the interface of the IFCS F-15 simulator developed by the WVU research team. The control framework of the simulator is based on the IFCS architecture, shown in Figure 1.2. The knowledge gained through the design and evaluation of the control schemes is of direct use in performance verification and validation.

Within the simulator, a non-linear approximate model of the F-15 aircraft for the simulation of a high performance military aircraft distributed by NASA to academic institutions within the 1990 AIAA GNC Design Challenge [77, 78, 79] is implemented. Primary control surfaces are shown in Figure 6.3. They are elevators, ailerons and rudders. Two types of control surface failure have been modeled, with the first type of failure corresponding to an

actuator failure resulting in a locked surface. In this scenario the user has the option to have the failed surface stay locked at the current position or in a pre-defined position. The second failure type corresponds to the case of a damaged control surface resulting in a drop in the efficiency of the control surface starting from the instant of failure. The user can set any individual of the control surfaces to fail: left or right stabilator, aileron, and rudder.

In reality, the flight envelope is very large. It is impossible to capture it using one support vector data description. Depending on different values of Mach (velocity) and altitude, the characteristics of nominal regions within the flight envelope could be wildly different. Based on domain expertise [78, 79], we divide the flight envelope into different sections. Figure 6.2 gives an illustration of the division. For example, in our experiments within section 1, the Mach is chosen from 0.6 to 0.7 and the altitude is simulated from 5000 to 5500 meters. Within each section, a boundary that separates the nominal region from off-nominal region is formed by running our fast SVDD algorithm. In operation, each incoming data frame is placed in a section and then tested using the corresponding SVDD boundary.

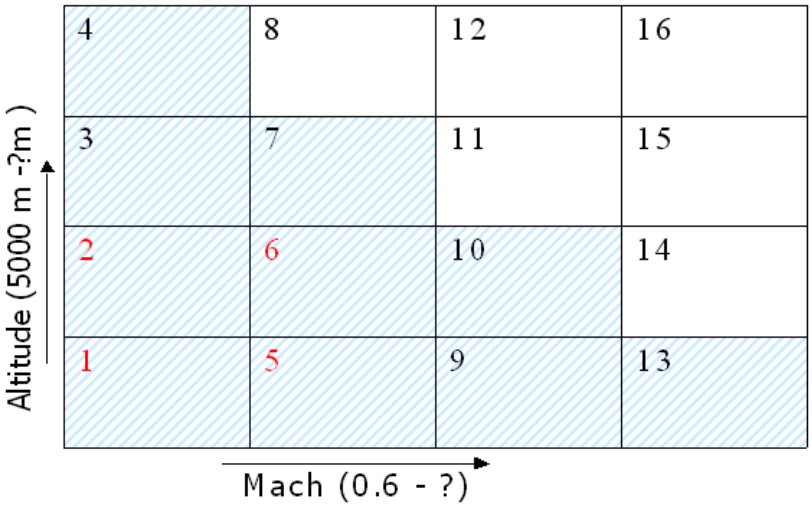


Figure 6.3: Flight section division of IFCS

We ran the simulator in the flight section 1 for 40 seconds at the simulation rate of 20Hz and collected 800 data points. We then simulated five different failure modes within the same section as follows.

1. **Mode 1 :** A control surface failure (locked left stabilator, stuck at 0 degree).
2. **Mode 2 :** A control surface failure (locked left stabilator, stuck at +3 degree).

3. **Mode 3 :** A control surface failure (locked left stabilator, stuck at -3 degree).
4. **Mode 4 :** A loss of control surface (50% missing surface of left stabilator).
5. **Mode 5 :** A loss of control surface (50% missing surface of right aileron).

Each failure mode simulation starts with a nominal flight. At the 600^{th} data point, a pre-defined failure is injected. These failure scenarios were simulated using the open-loop simulation. This means that the compensations generated by the online neural network have not been propagated into the controller. The chosen simulation data depict off-nominal flight conditions of approximately 10 seconds of flying time, corresponding to 200 frames of data at the simulation rate of 20Hz. A data frame is a point in a six-dimensional space corresponding to four sensor readings (independent variables) and two stability and control derivative errors from PID and PTNN (dependent variables). The online neural network tested here is the $DCS - C_z$ network, one of the five DCS sub-networks of the IFCS. The independent variables are Mach number (the ratio of the speed of the aircraft to the local speed of sound), α (aircraft's angle of attack), β (the pitch rate of the aircraft), and altitude of the aircraft. The dependent variables are two stability and control derivative errors generated by the difference between PID and PTNN, which are $\Delta Cz\alpha$ and $\Delta Cz\beta$. When the DCS network is in recall, it predicts these two derivative corrections.

We need to emphasize that in our experiments, scaling is applied prior to DCS training and SVDD in order to avoid unnecessary loss of dimensionality. We transform the data set by subtracting the column mean from each column and dividing each column by its standard deviation. After the standardization, every column has mean 0 and standard deviation 1.

6.2 Failure Detection using SVDD

The evaluation of our fast SVDD method consists of two steps. In the first step, we verify the correctness of our fast SVDD solution by comparing it with the original solution. Then, we test the boundary using these five failure mode data.

6.2.1 Comparison of the two SVDD algorithms

We chose α and $\Delta Cz\alpha$ as dimensions for the two-dimensional demonstration. We ran both the original and the fast SVDD algorithms on an identical data set and recorded their execution time. Normalization was performed on the data set. However, no particular speculation was made concerning the structure of the data. Figure 6.4 presents the results of

running the original SVDD algorithm and our fast SVDD algorithm, shown as the left plot and the right plot, respectively. In both plots, 800 nominal flight data frames are represented by crosses, with x-axis representing α and y-axis representing $\Delta Cz\alpha$. The boundaries are shown as lines circulating the data points. In Figure 6.4, the plot on the right illustrates the last step of our fast SVDD algorithm. In the final merging step, these two boundaries are represented by blue line and red line. The final form of description is represented by the black line. All data points inside the final boundaries are considered nominal data. Evidently, these two final data description forms in Figure 6.4 are identical, demonstrating the capability of our fast SVDD algorithm (described in Chapter 4) to reach a global solution.

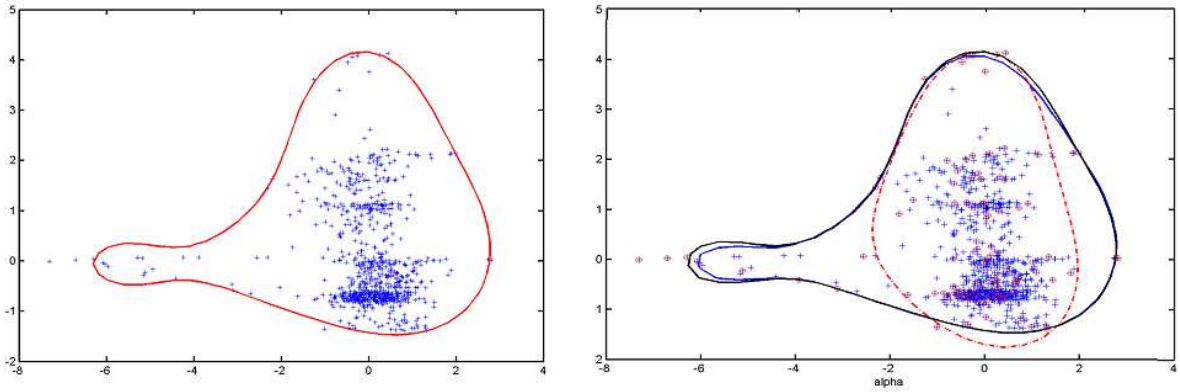
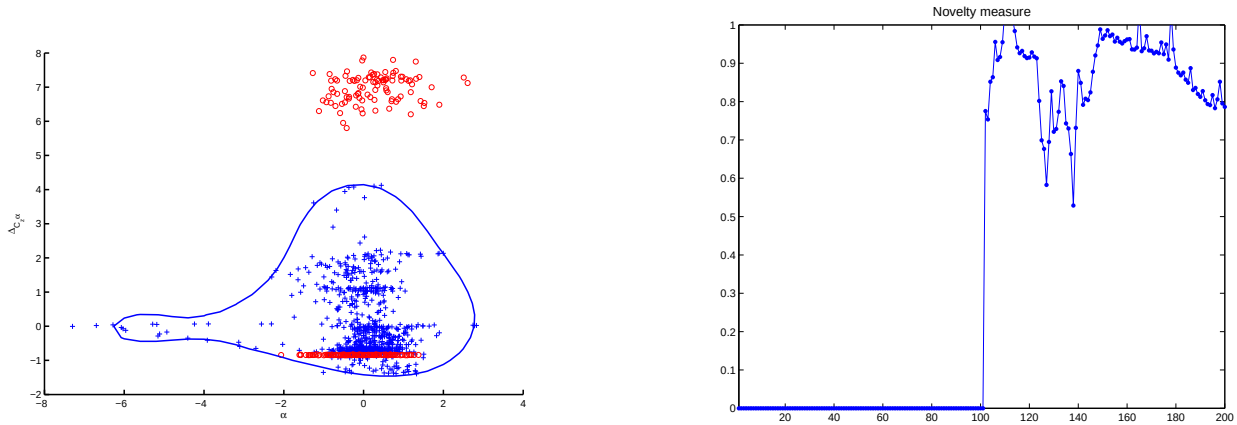


Figure 6.4: Compare SVDD results of two algorithms running on flight control simulation data of size 800. Left: SVDD using original SVDD algorithm. Right: the "decomposing and combining" SVDD algorithm.

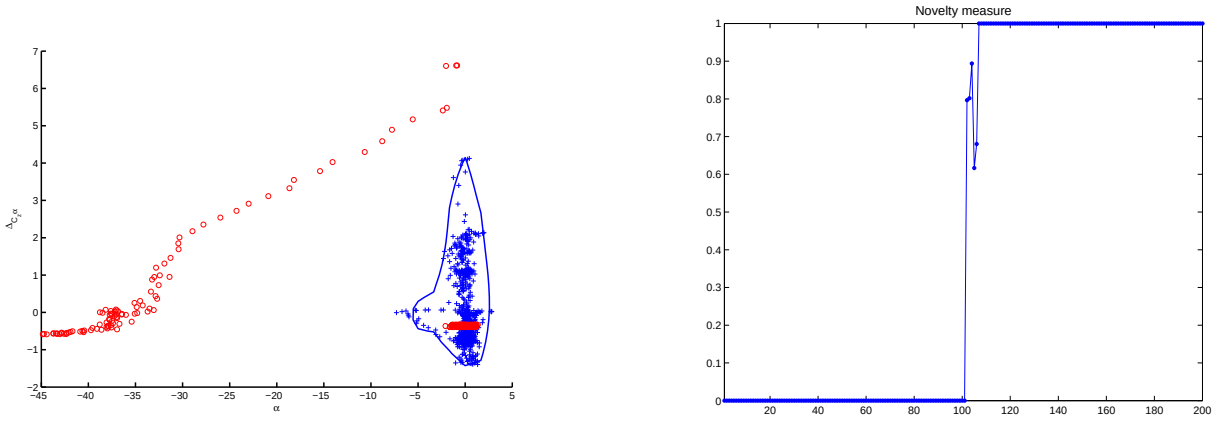
6.2.2 Failure Detection Using Fast SVDD

We tested the novelty detection capability of the SVDD method using five failure mode data. Probabilistic measures are computed as an indicator of how novel a particular data point is from the learned domain. Figures 6.5-6.9 present the results of failure detection of these five failure modes, respectively. In Each figure, Plot (a) shows the two-dimensional demonstration of the detection results, where the data points originating from a failure mode is represented by circles and the nominal data points are represented by crosses. Any data point that falls outside the SVDD boundary (shown as black line), is considered novel. Plot (b) shows the real-time novelty measures for each of five failure mode simulations in a



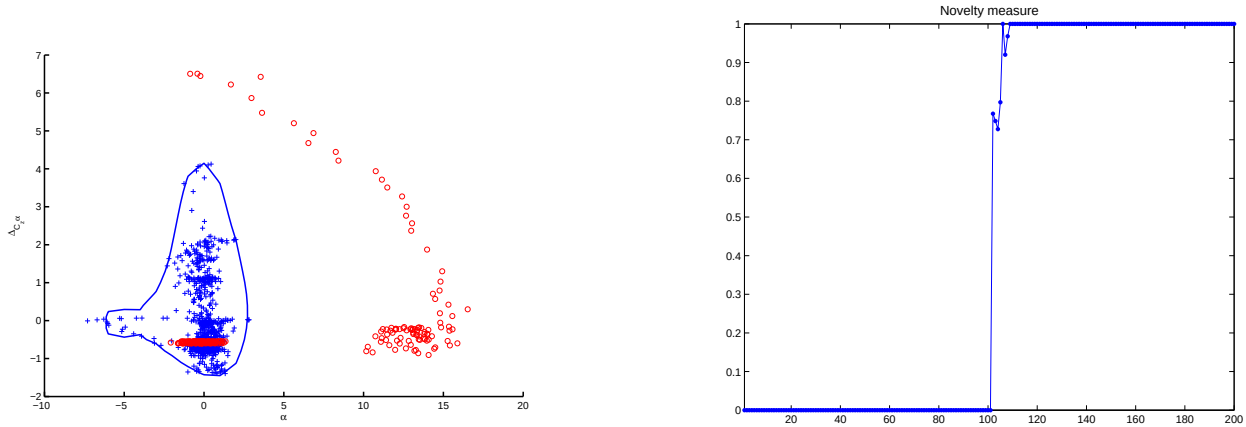
(a) **(b)**

Figure 6.5: Novelty detection results using SVDD on Failure Mode 1 simulation data. (a): SVDD of nominal flight simulation data is used to detect novelties. (b): Novelty measures returned by SVDD tool for each testing data point.



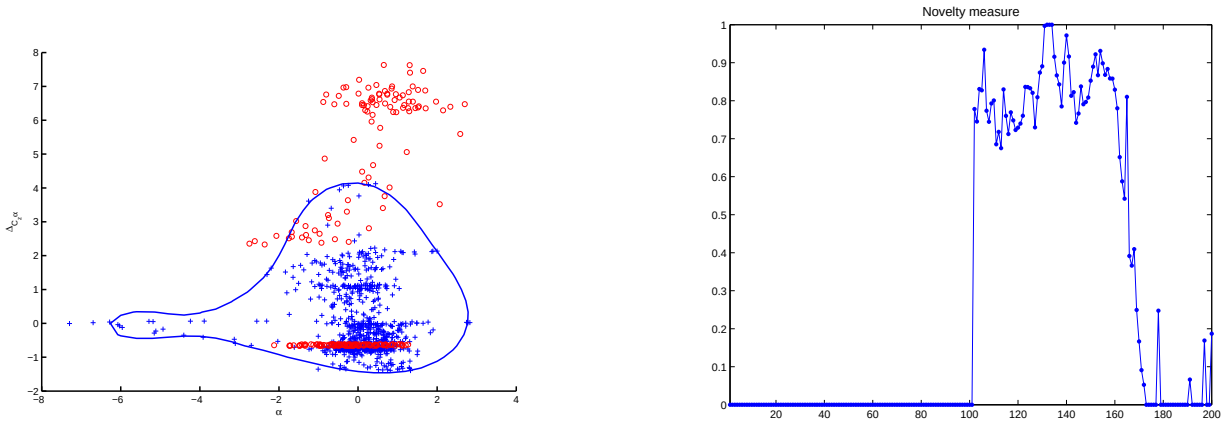
(a) **(b)**

Figure 6.6: Novelty detection results using SVDD on Failure Mode 2 simulation data. (a): SVDD of nominal flight simulation data is used to detect novelties. (b): Novelty measures returned by SVDD tool for each testing data point.



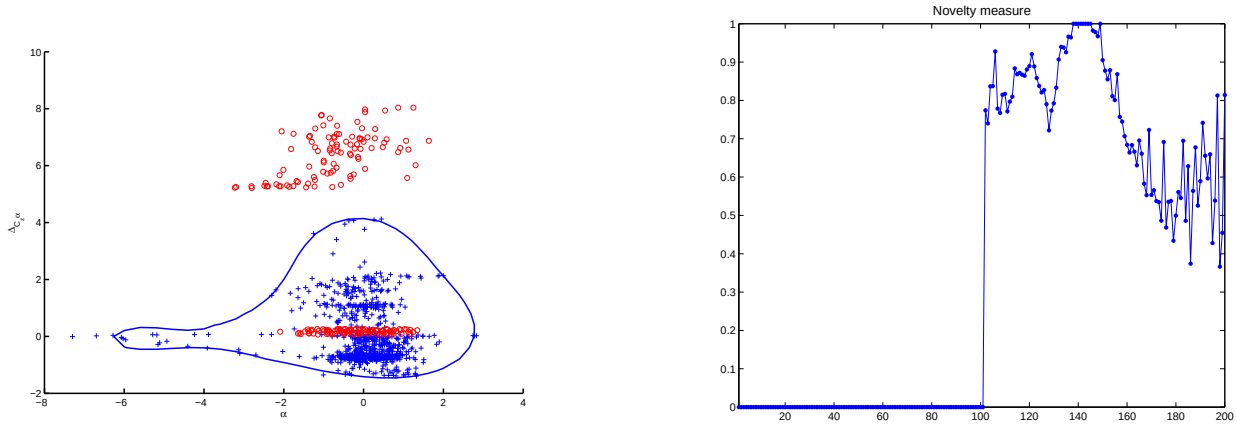
(a) **(b)**

Figure 6.7: Novelty detection results using SVDD on Failure Mode 3 simulation data. (a): SVDD of nominal flight simulation data is used to detect novelties. (b): Novelty measures returned by SVDD tool for each testing data point.



(a) **(b)**

Figure 6.8: Novelty detection results using SVDD on Failure Mode 4 simulation data. (a): SVDD of nominal flight simulation data is used to detect novelties. (b): Novelty measures returned by SVDD tool for each testing data point.



(a) **(b)**

Figure 6.9: Novelty detection results using SVDD on Failure Mode 5 simulation data. (a): SVDD of nominal flight simulation data is used to detect novelties. (b): Novelty measures returned by SVDD tool for each testing data point.

time sequence. As we can see, all plots demonstrate the highest degree of novelty after the 100^{th} data frame, where the simulated failures occur. Such measures can be interpreted as distance between an incoming data point and the pre-defined safe operational region, shown in corresponding figures.

Failure Mode 2 and Failure Mode 3 are two control surface failures with stabilator stuck at a certain degree (+3 and -3). They indicate a more substantial damage than the loss-of-surface type of failure. Comparing the novelty measures shown in Figure 6.6 and Figure 6.7 with those shown in Figure 6.5, Figure 6.8 and Figure 6.9, we can see the results evidently show that the degree of novelty of these two failures are much higher than those of other failures. In the plot of Figure 6.6(a) and Figure 6.7, depicting the stuck-at control surface failures, a large portion of failure mode data falls outside the boundary. Moreover, the data points in both figures fall further outside the nominal data boundary than the data points in Figure 6.5(a), Figure 6.8(a) and Figure 6.9(a). Correspondingly, in plots of Figure 6.6(b) and Figure 6.7(b), we can see that the novelty measures of stuck-at control surface failure data after 100^{th} data frame are larger than those in other failure data. In all five figures, after the 100^{th} point, when failures occurred, SVDD detects the abnormal changes and returns with the highest novelty measures. This demonstrates the reasonably effective and accurate detection capabilities of our novelty detector using SVDD. Moreover, compensated with the

stability measures from online monitors, they can be further integrated into inferences, upon which decisions about trustworthiness of outputs generated by adaptive component can be made.

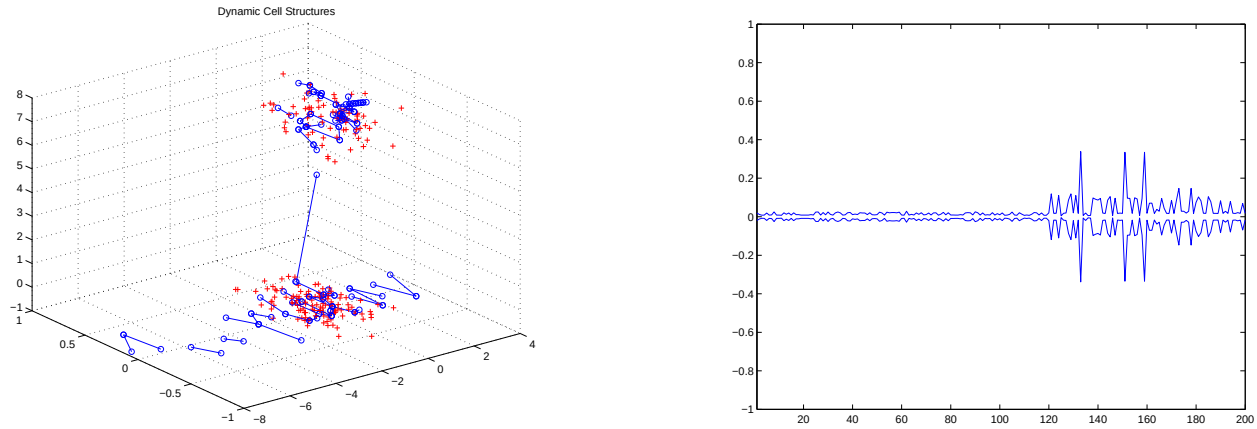
6.3 Validity Index in DCS

The online neural networks in IFCS learn on the environmental changes and accommodate failures. They generate derivative corrections as compensation to the PTNN output (see Figure 1.2). We conducted two sets of experiments to evaluate the accommodation performance and validate the predictions of the DCS network using validity index. We first simulate the online learning of a DCS network on different failure modes and compute the validity index in real-time to evaluate the adaptability of the DCS network in terms of how well and how fast the DCS network accommodates the failures. We then use the validity index to evaluate the adaptability of DCS in terms of comprehensiveness to check how the DCS network responds to other failure conditions after it has accommodated one failure condition. We carry out the first experimentation in real-time simulation. In the second experimentation, we use the DCS network as an off-line predictor after it has learned on a particular failure.

6.3.1 Online Testing of Validity Index

In this experimentation, we simulate the online learning of the DCS network under five different failure mode conditions. Running at 20 Hz, the DCS network updates its learning data buffer at every second and learns on the up-to-date data set of size 200. In each run, we first start the DCS network under nominal flight conditions with 200 data points. After that, every second, we first set the DCS network in recall mode and calculate the derivative corrections for the freshly generated 20 data points, as well as their validity index. Then we set the DCS network back into the learning mode and update the data buffer. While updating the data buffer, we discard the first incoming 20 data points and add the freshly generated 20 data points to maintain the buffer size, i.e., 200. The DCS network continues learning and repeats the recall-learn procedure.

Figure 6.10, Figure 6.11, Figure 6.12, Figure 6.13, and Figure 6.14 show the experimental results of the simulations on failure Mode 1, Mode 2, Mode 3, Mode 4 and Mode 5, respectively. In each figure, Plot (a) shows the final form of the DCS network structure at the end of the simulation. As a three-dimensional demonstration, the x-axis and y-axis represent these two independent variables, α and β, respectively. The z-axis represents one derivative

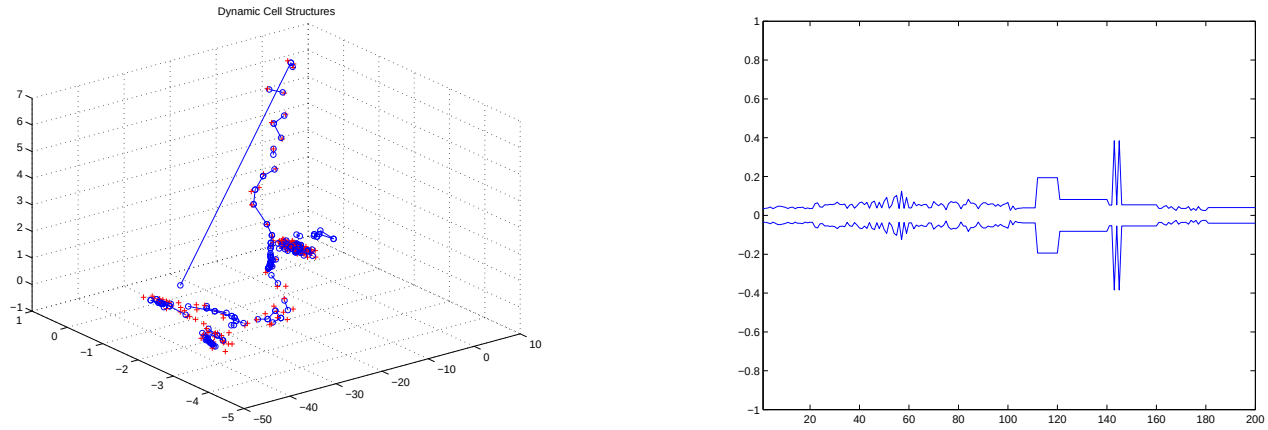


(a) **(b)**

Figure 6.10: Testing on Failure Mode 1 simulation data in real-time (running at 20Hz, failure occurs at 100^{th} data frame). (a): The final form of DCS network structures. (b): Validity Index shown as error bars for each DCS output.

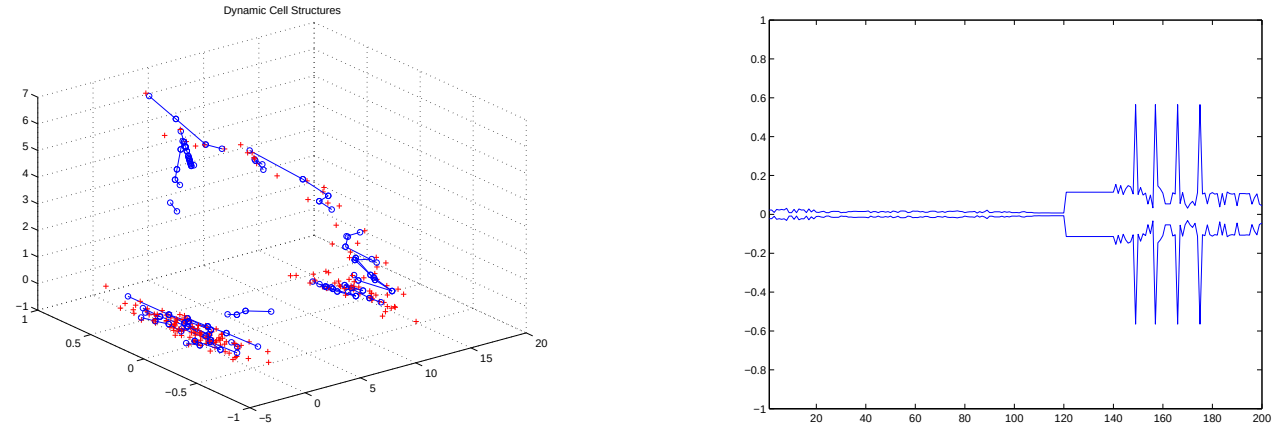
correction, $\Delta Cz\alpha$. The 200 data points in the data buffer at the end of the simulation are shown as red crosses in the 3-D space. The network structure is represented by blue circles (as neurons) connected by lines as a topological mapping to the learning data. Plot (b) presents the validity index, shown as error bars. The x-axis here represents the time frames. The failure occurs at the 100^{th} data frame. We compute the validity index for the data points that are generated five seconds before and five seconds after the failure occurs. In total, Plot (b) illustrates the validity index for 200 data points.

A common trend revealed by the validity index in all five simulations is the increasingly larger error bars after the failure occurs. Then, the error bars start shrinking while the DCS network starts adapting to the new domain and accommodating the failure conditions. After the failure occurs, the change (increase/decrease) of the validity index varies. This depends on the degree of severeness as well as the accommodation performance of the DCS network. For instance, we spot a better accommodation performance under failure Mode 1 than failure Mode 2 and Mode 3, shown by the narrower error bars and faster shrinkage of the after-failure error bars. This is consistent with the fact the failure conditions under Mode 2 or Mode 3 are more severe than the conditions under failure Mode 1. In all scenarios, the validity index explicitly indicates how well and how fast the DCS network accommodates the failures.



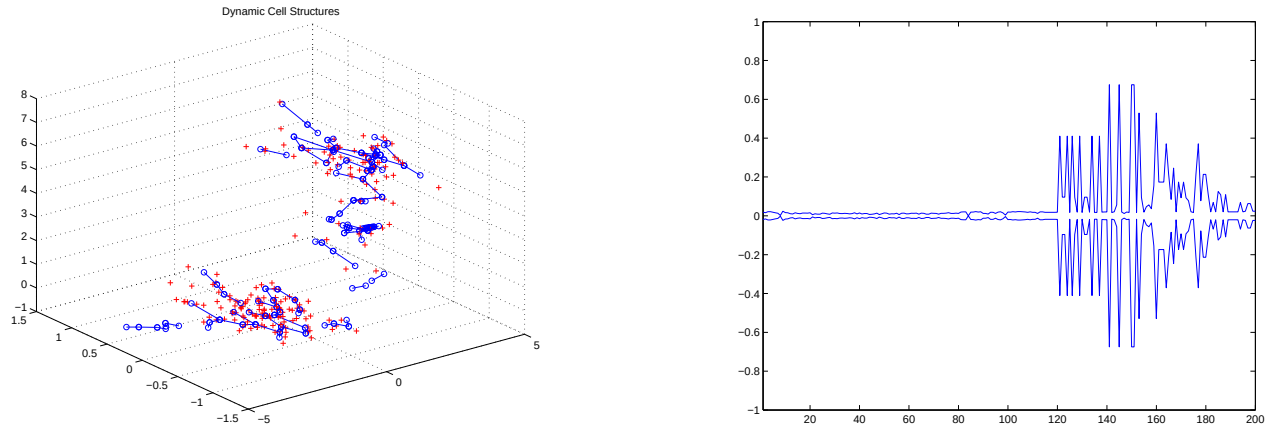
(a) **(b)**

Figure 6.11: Testing on Failure Mode 2 simulation data in real-time (running at 20Hz, failure occurs at 100^{th} data frame). (a): The final form of DCS network structures. (b): Validity Index shown as error bars for each DCS output.



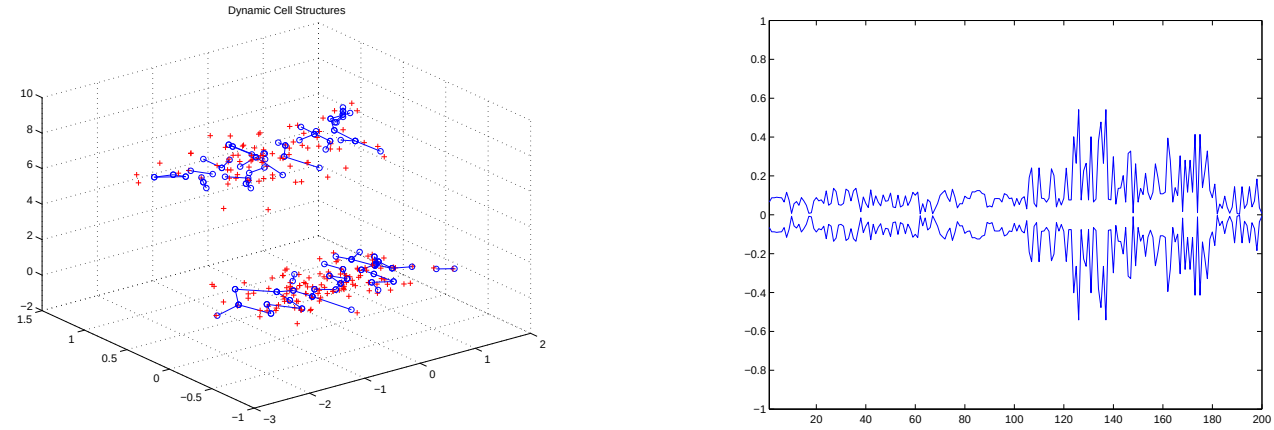
(a) **(b)**

Figure 6.12: Testing on Failure Mode 3 simulation data in real-time (running at 20Hz, failure occurs at 100^{th} data frame). (a): The final form of DCS network structures. (b): Validity Index shown as error bars for each DCS output.



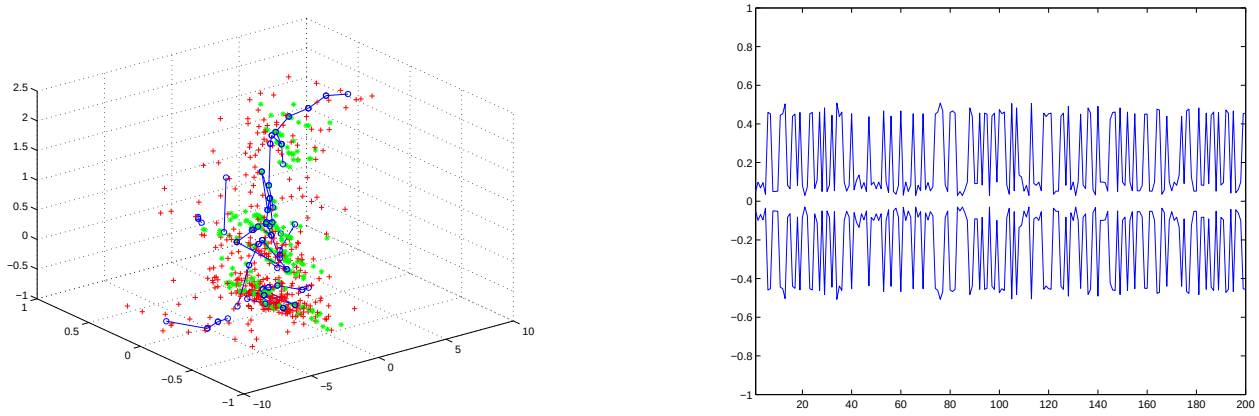
(a) **(b)**

Figure 6.13: Testing on Failure Mode 4 simulation data in real-time (running at 20Hz, failure occurs at 100^{th} data frame). (a): The final form of DCS network structures. (b): Validity Index shown as error bars for each DCS output.



(a) **(b)**

Figure 6.14: Testing on Failure Mode 1 simulation data in real-time (running at 20Hz, failure occurs at 100^{th} data frame). (a): The final form of DCS network structures. (b): Validity Index shown as error bars for each DCS output.



(a) **(b)**

Figure 6.15: Testing on Failure Mode 1 simulation data after the network learns till training error<0.1. (a): DCS network structures and predictions. (b): Validity Index shown as error bars for each DCS output.

6.3.2 Off-line Testing of Validity Index

In the off-line experiments, the DCS network first learns on one failure mode, Mode 1 in particular. After learning for 20 seconds, the DCS network functions in the recall mode as a parameter predictor. The network is then tested with all five failure modes. In our experiments, the DCS network predicts the value of $\Delta Cz\alpha$, one of the critical derivative corrections. Each failure mode corresponds to 200 data frames. We inject failure flight conditions at the 100^{th} data frame. The test data is then run through the DCS network. The DCS predicts the $\Delta Cz\alpha$ values and computes the validity index for each output as we depict it in form of the error bars.

A low confidence interval can be caused by either inadequate learning or lack of training data in particular regions. In our first experiment, the DCS network stops learning at different stopping criteria. The DCS network shown in Figure 6.15 stops earlier than the network shown in Figure 6.16. Thus, the latter has been trained more adequately than the former. Both are trained on the same data set using the Failure Mode 1 data and tested using the same data set. The predictions are shown in both Figure 6.15(a) and Figure 6.16(a) as green asterisks. The final forms of the neural network structure are represented by blue circles, representing neurons connected by lines. The validity index is shown as error bars in Figure 6.15(b) and Figure 6.16(b). We can see that, with much narrower error bars, the

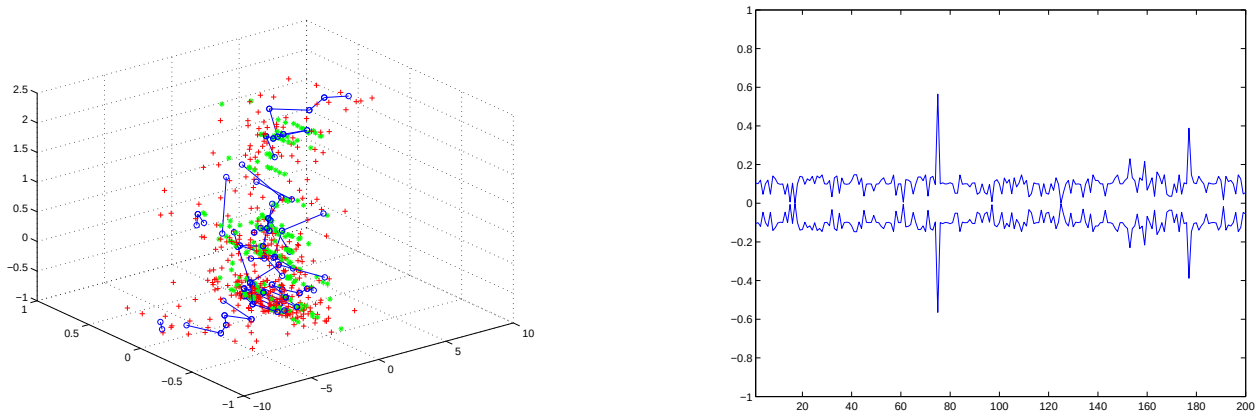
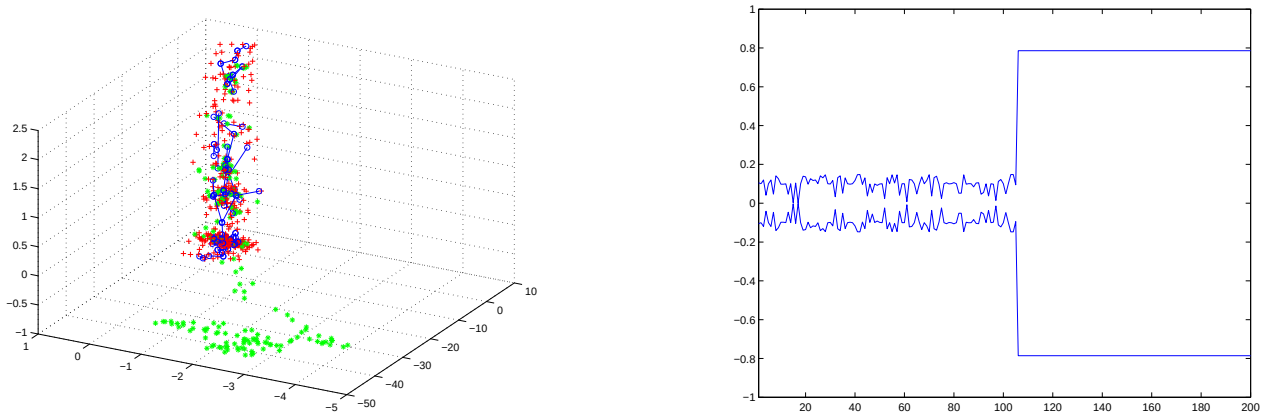


Figure 6.16: Testing on Failure Mode 1 simulation data after the network learns till training error$<$ 0.05. (a): DCS network structures and predictions. (b): Validity Index shown as error bars for each DCS output.

validity index shown in Figure 6.16(b) demonstrates a better prediction performance than that shown in Figure 6.15(b).

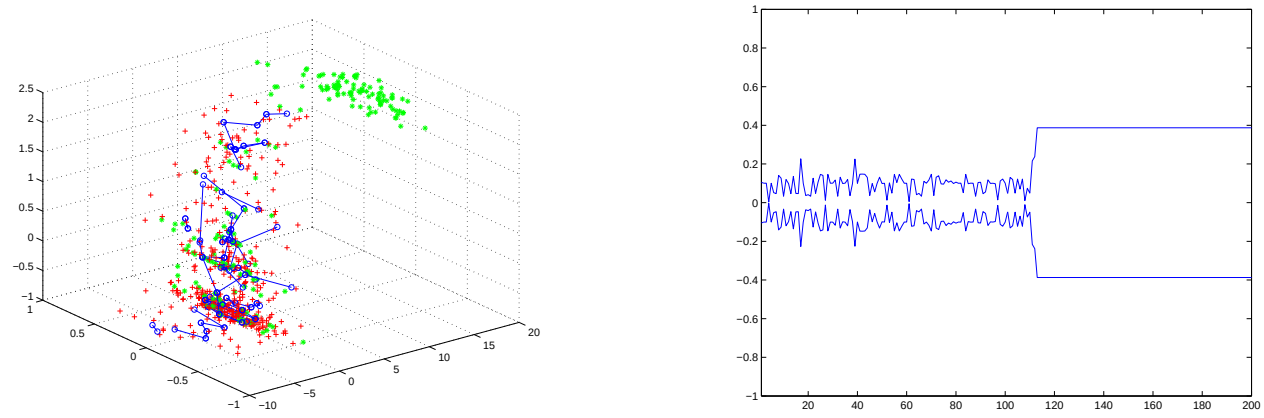
We then use the DCS network shown in Figure 6.16 to test on all other four failure modes. Figure 6.17, Figure 6.18, Figure 6.19 and Figure 6.20 present the experimental results for Failure Mode 2, Mode 3, Mode 4 and Mode 5, respectively. In each figure, Plot (a) illustrates the final form of the DCS network. The training data points are represented by the red crosses. The neurons represented by the blue circles are connected by lines. The predictions generated by the network along with the input values (two dimensions, α and β) are shown as green asterisks in the 3-D space. In Plot (b)'s, we show the validity index as error bars for all 200 data frames in a time sequence.

As we can see from all four figures, as soon as the test data points start to deviate from the learned domain, the error bars associated with the output increase. A large error bar indicates that the degree of uncertainty reflected by the validity index for this particular output is relatively high. In this instance, further actions can be taken to verify the correctness of such an output. Furthermore, we can identify from the results that accommodation to Failure Mode 1 does not necessarily give the DCS network the ability to accommodate to other failures. Particularly, the error bars in Figure 6.17 (b) and Figure 6.18 (b) indicate a relatively low confidence level for predictions under both conditions. This is consistent with the fact that Failure Mode 1 and Failure Mode 2 are two failure modes simulating a rela-



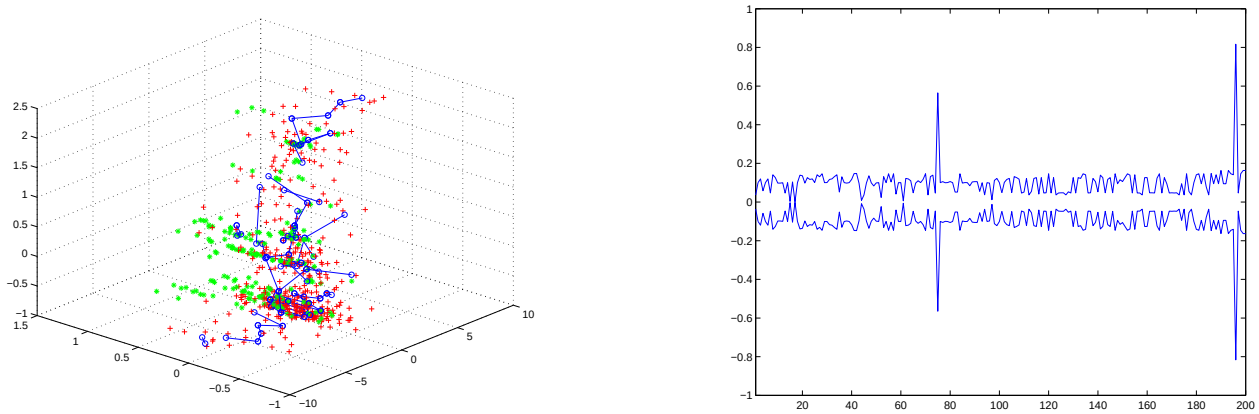
(a) **(b)**

Figure 6.17: Testing on Failure Mode 2 simulation data after the network accommodates Failure Mode 1. (a): DCS network structures and predictions. (b): Validity Index shown as error bars for each DCS output.



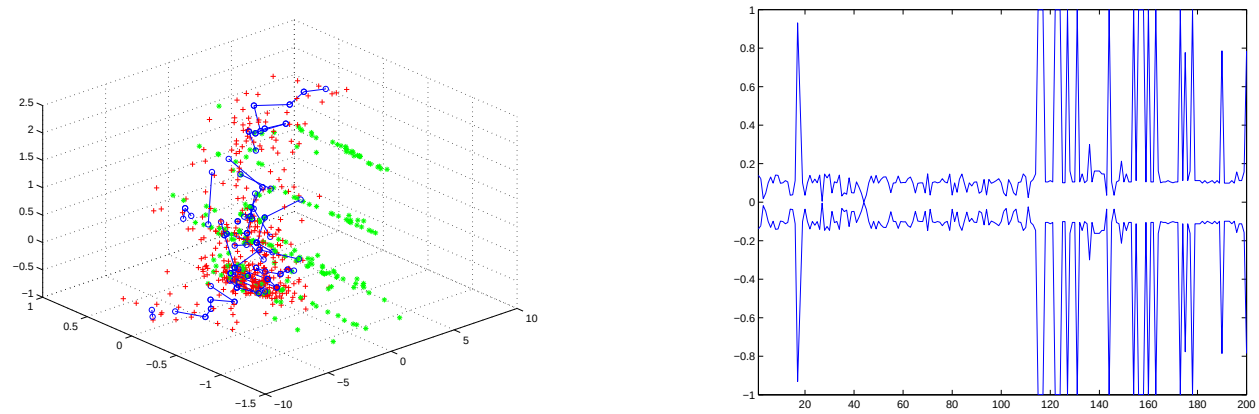
(a) **(b)**

Figure 6.18: Testing on Failure Mode 3 simulation data after the network accommodates Failure Mode 1. (a): DCS network structures and predictions. (b): Validity Index shown as error bars for each DCS output.



(a) **(b)**

Figure 6.19: Testing on Failure Mode 4 simulation data after the network accommodates Failure Mode 1. (a): DCS network structures and predictions. (b): Validity Index shown as error bars for each DCS output.



(a) **(b)**

Figure 6.20: Testing on Failure Mode 5 simulation data after the network accommodates Failure Mode 1. (a): DCS network structures and predictions. (b): Validity Index shown as error bars for each DCS output.

tively severe type of control failures, which causes more control damages and is significantly different from the moderate type of failures such as Failure Mode 1, Failure Mode 4 and Failure Mode 5.

6.4 Summary

Through the experiments testing our fast SVDD tool, we successfully demonstrate the consistency between the data description of our fast algorithm and that of the original algorithm and verify the correctness of the fast SVDD algorithm. Meanwhile, the efficiency of training is improved. The running time of our fast SVDD algorithm runs as much as 100 times faster than the original algorithm on a data set of size 800.

The experimental results of failure detections on five different failure modes of IFCS prove that the SVDD tool is effective and accurate in detecting failures. Since the evaluation of testing data points only involves "support vectors", usually a relatively small fraction of the data set, the detection of novelties is computationally efficient.

The validity index in DCS is tested online to evaluate the accommodation performance of the DCS network. Five failure modes are tested. Under the recall mode, the validity index is generated for each testing data point right after the DCS learns. In these five tests, a larger validity index explicitly indicates the poor prediction performance after failure occurs. More importantly, it effectively demonstrates how well and how fast the DCS network adapts to a failure condition and tries to accommodate the failure.

We also use validity index to test the comprehensiveness of the DCS network's accommodation to failures. Four failure modes are used to test the DCS network after the network has adapted to one of the moderate failure mode conditions. The computed validity index effectively indicates poor prediction performance for more severe failure mode conditions. Meanwhile, the validity index also reflects lower reliability within regions characterized by sparse data. This demonstrates that the validity index in DCS is an effective approach and can be applied to validate the DCS prediction performance after adaptation.

Having improved the efficiency of the SVDD algorithm, we can integrate our novelty detectors with online stability monitors, the second step of proposed V&V approach (See Chapter 3) to validate the failure accommodation capabilities of the DCS networks. When DCS is used for prediction, we can perform validity checks based on the validity index to verify the correctness of the DCS output and further evaluate the accommodation performance at the system output layer. Together, they can be deployed within the context of the adaptive flight control system architecture, providing an in-flight V&V capability.

Chapter 7

Conclusions and Future Work

Online Adaptive Systems are systems whose function evolves over time, as they improve their performance through online learning. The advantage of adaptive systems is that they can, through judicious learning, react to situations that were never individually identified and/or analyzed by the designer. In recent years NASA conducted experiments evaluating adaptive computational paradigms (neural networks, AI planners) for providing fault tolerance capabilities in control systems following sensor and/or actuator faults. Experimental success suggests significant potential for future use. The critical factor limiting wider use of online adaptive systems in process control applications is our (in)ability to provide a sound and practical approach to their verification and validation.

For all their qualities, adaptive systems are inherently difficult to verify/ validate, precisely because they are adaptive. Research work has focused on system evaluation through testing. Popular methods such as cross-validation during training, bias-variance trade-off, etc., are favorite approaches for balancing the memorization and generalization abilities. However, checking all possible inputs is impossible. The real-time constraints of online adaptive systems take the problem to a harder level. Investigation in V&V of neural networks shows that the online learning behavior of a neural network model is very complex and thus nearly impossible to be verified using a single type of off-line analytical method. Moreover, conventional V&V techniques are generally static and thus useless for such systems due to the continual changes of the environment.

Having investigated the related work done towards V&V of online adaptive systems and neural network models, we sought solutions to the following problems.

1. Can we develop a dynamic validation approach for neural network-based online adaptive systems that meets the requirement of practicality, reliability and efficiency?

2. Since the embedded neural network models learn in an online fashion, their functionality is subject to the learning data and their performance is particularly vulnerable to failure conditions. Therefore, at the input layer of an online adaptive system, detecting the potential failures becomes very important in the context of online validation. The problem is - how to detect and diagnose the failure conditions accurately and efficiently for an online adaptive system?

3. A reliable neural network model should not only provide robust learning performance but also predictions with confidence. Poor prediction performance of a neural network caused by either lack of data or inadequate learning brings serious concerns on the validity of the output. This occurs at the system output layer, which is the final phase of an online adaptive system after the fault accommodation. It is very important for us to be able to check the validity of the network predictions. Hence, in this respect, our problem is - how to define, quantify and efficiently compute the confidence for the network predictions in an online adaptive system?

7.1 Achievements

In this thesis, we investigated the V&V problem of neural network-based online adaptive systems and proposed a dynamic validation approach using two real-time validation techniques. The major achievement we accomplished in our research is a feasible and robust validation scheme for a neural network-based online adaptive system.

7.1.1 A Validation Framework

We developed a non-conventional approach for validating the performance adequacy of a neural network-based online adaptive system. The proposed framework combines several methodologies, as presented in Chapter 3. It consists of three important steps with each step in charge of a significant feature of the online adaptive learner.

1. At the input layer, an independent novelty detector scrutinizes the incoming data stream (the input) for the adaptive learner, detects failures and alerts when potentially hazardous data are present.

2. During adaptation, the run-time stability monitors inspect the learning behavior and inherent functionality changes of the adaptive learner.

3. At the output layer, when the adaptive learner is employed for prediction, a validity check is imposed on each output generated by the learner. Based on system safety requirements, it discovers the output whose confidence measure is low enough to be regarded "invalid". These invalid outputs will then be discarded and the propagation of unsafe data/signals into the next component (for example, a controller) will be disallowed.

At every stage, domain knowledge can be inferred from the results provided by the tools and proper actions will be taken accordingly.

7.1.2 Validation Methods

As a parallel research effort, the run-time stability monitors are made available by other researchers at WVU [8]. Our research focuses on developing the validation techniques at both the input and the output layers. In Chapter 2, we investigated validation approaches for neural network models and novelty detection techniques in the literature. Based on our studies, we proposed two different approaches, the Support Vector Data Description (SVDD) and the validity index in dynamic cell structure networks, as validation methods deployed at the input layer and the output layer respectively to validate a typical online neural network-based adaptive system, the Intelligent Flight Control System (IFCS).

At the input layer, we adopted the support vector data description as our novelty detection tool. As a one-class classifier, SVDD is a very promising tool for novelty detection. However, due to space complexity of matrix operations, the optimization process becomes memory and time consuming when n, the size of training set increases. We developed an algorithm that first reduces the space complexity by breaking the training data set into subsets at random and apply SVDD to each subset. Then, based on two lemmas of random sampling and SVDD combining, we merge the data descriptions into a "common decision boundary". Provided with the fact that usually the number of support vectors is relatively few with respect to n, the search for a common decision boundary for a training data set of size n in a $d-$dimension space can be bounded at $O(d^{\frac{3}{2}} n^{\frac{3}{2}} \log n)$ steps. Experiments have been conducted and results are compared with original solution. We demonstrated the efficiency as well as the effectiveness of the improved algorithm.

At the output layer, we developed a statistical method to validate the neural network predictions of the IFCS. We define a reliability-like measure, the validity index, for the Dynamic Cell Structure (DCS) network used for adaptive learning in IFCS. The validity index is computed based on the statistical information collected through the learning process and

computed as a confidence interval for each output. It reflects the reliability level associated with each output and thus can be used to perform validity checks and verify the accommodation performance of the online adaptive component. The computation of validity index is simple and requires no additional learning. Experimental results demonstrated that the developed techniques effectively detect system failures and provide validation inferences in a real-time manner.

7.2 Conclusions

We developed a non-conventional approach for validating the performance of a neural network based online adaptive system. The validation framework consists of

- Independent failure detections at the input layer and validity checks at the output layer that provide validation inferences for verifying the accommodation capabilities of the online adaptive component in the context of fault tolerance, and
- Runtime stability monitors that examine the stability properties of the neural network adaptation.

We developed validation techniques to examine: 1)the learning data on which the online adaptive component is trained, and 2) the neural network predictions after the adaptation. At the input layer, an independent online failure detection technique is implemented before the data propagate into the learner. At the output layer, the learner produces output in response to the environmental changes. An online reliability measure associated with each neural network prediction is generated for validity check. Experimental results using the data collected through an F-15 aircraft IFCS simulator show that:

1. SVDD can be adopted for defining nominal performance regions for the given application domain and thus used for failure detection. By improving its computational efficiency, our fast SVDD algorithm achieves the ability to provide successfully automated separation between potential failures and normal system events in real-time operation.
2. The validity index in DCS is an improvement to DCS network models. The implementation of validity index is straightforward and computationally efficient. As a confidence measure for DCS output, the validity index can effectively indicate poor fitting within regions characterized by sparse data and/or inadequate learning. It is a

feasible approach for evaluating the accommodation performance of the DCS network at the system output layer.

The validation methods at both input and output layers provide the bases for the effective validation of the IFCS as a typical example of a neural network- based online adaptive system. In [8, 10], the online stability monitors have shown a successful realization of convergence tracking of adaptation error towards a stable (or unstable) and safe (or unsafe) state in the adaptive flight control system. The successful applications of these methods complete a practical, reliable and computationally efficient validation framework.

We conclude that the proposed methodology provides a promising approach for validating neural network-based online adaptive systems. The observed efficiency and scalability of both methods give us the expectation that the proposed V&V method can be successfully applied to other types of online adaptive learners.

7.3 Future Work

Experiments on IFCS have demonstrated the feasibility of the proposed validation approach with successful applications of the developed methodologies. We plan to proceed with investigations in this direction by applying these sophisticated techniques. In the future, we will extend our work as follows.

1. Further our investigation in validation at both layers by completing the flight envelope for IFCS. Furthermore, within each flight section, we will derive reasonable thresholds for failure detection using flight simulation data.

2. We will implement both methods within the simulator and integrate with the online stability monitors to provide forward and backward recovery capabilities for IFCS.

3. Computational efficiency and scalability of our methods give us confidence that the proposed V&V method can be generalized to other adaptive applications. Hence, in the following study, we will continue tuning our validation framework for extensive neural network-based online adaptive systems.

Bibliography

[1] Institute for Telecommunication Sciences, Definition of Adaptive Systems. `http://glossary.its.bldrdoc.gov/fs-1037/dir-002/_0154.htm`.

[2] A. Mili, B. Cukic, Y. Liu, and R. Ben Ayed. Towards the Verification and Validation of On-Line Learning Adaptive Systems, In *Computational Methods in Software Engineering*. Kluwer Scientific Publishing, 2003.

[3] C. C. Jorgensen. Feedback linearized aircraft control using dynamic cell structures, *World Automation Congress* (ISSCI), Alaska, 1991, 050.1-050.6.

[4] The Boeing Company, Intelligent flight control: advanced concept program, *Technical report*, 1999.

[5] M.A. Boyd, J. Schumann, G. Brat, D. Giannakopoulou, B. Cukic and A. Mili. Validation and verification process guide for software and neural nets. Technical report, NASA Ames Research Center, September 2001.

[6] J. Schumann, and S. Nelson, Towards V&V of neural network based controllers. *Workshop on Self-Healing Systems*, 2002.

[7] D. Mackall, S. Nelson, and J. Schumann. Verification and validation of neural networks of aerospace applications, Technical report, CR-211409, NASA, 2002.

[8] S. Yerramalla, B. Cukic and E. Fuller. Lyapunov Stability Analysis of Quantization Error for DCS Neural Networks. *Int'l Joint Conference on Neural Networks (IJCNN'03)*, Oregon, July 2003.

[9] Y. Liu, S. Yerramalla, E. Fuller, B. Cukic and S. Gururajan. Adaptive Control Software: Can We Guarantee Safety? *Proc. of the* 28^{th} *International Computer Software and Applications Conference, workshop on Software Cybernetics*, Hong Kong, September 2004.

[10] S. Yerramalla, Y. Liu, E. Fuller, B. Cukic and S. Gururajan. An Approach to V&V of Embedded Adaptive Systems, in *Lecture Notes in Computer Science (LNCS) Proceeding of Third NASA-Goddard/IEEE Workshop on Formal Approaches to Agent-Based Systems*, Springer-Verlag, 2004.

[11] Ch. Brink,W. Kahl, and G. Schmidt. Relational Methods in Computer Science. *Springer Verlag*, New York, NY and Heidelberg, Germany, 1997.

[12] Ch. Alexander, D. DelGobbo, V. Cortellessa, A. Mili, and M. Napolitano. Modeling the fault tolerant capability of a flight control system: An exercise in SCR specifications. In *Proceedings, Langley Formal Methods Conference*, Hampton, VA, June 2000.

[13] E.W. Dijkstra. A Discipline of Programming. *Prentice Hall*, 1976.

[14] C.C. Morgan. *Programming from Specifications*. International Series in Computer Sciences. Prentice Hall, London, UK, 1998.

[15] J.R. Abrial. *The B Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[16] Z. Manna. *A Mathematical Theory of Computation*. McGraw Hill, 1974.

[17] H.D. Mills, V.R. Basili, J.D. Gannon, and D.R. Hamlet. *Structured Programming: A Mathematical Approach*. Allyn and Bacon, Boston, Ma, 1986.

[18] J. Dean. Timing the testing of cots software products. In *First International ICSE Workshop on Testing Distributed Component Based Systems*, Los Angeles, CA, May 1999.

[19] H. Hecht, M. Hecht, and D. Wallace. Toward more effective testing for high assurance systems. In *Proceedings of the 2nd IEEE High Assurance Systems Engineering Workshop*, Washington, D.C., August 1997.

[20] M. Lowry, M. Boyd, and D. Kulkarni. Towards a theory for integration of mathematical verification and empirical testing. In *Proceedings, 13th IEEE International Conference on Automated Software Engineering*, pages 322C 331, Honolulu, HI, October 1998. IEEE Computer Society.

[21] H. Ammar, B. Cukic, C. Fuhrman, and A. Mili. A comparative analysis of hardware and software Impact on software reliability engineering. *Annals of Software Engineering*, 10, 2000.

[22] D. K. Pradhan. Fault Tolerant Computing: Theory and Practice. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[23] B. Randall. System structure for software fault tolerance. IEEE Transactions on Software Engineering, SE-1(2), 1975.

[24] D.P. Siewiorek and R. S. Swarz. The Theory and Practice of Reliable System Design. Digital Press, Bedford, Mass, 1982.

[25] A. Avizienis. The n-version approach to fault tolerant software. *IEEE Trans. on Software Engineering*, 11(12), December 1985.

[26] K.M. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (1989) 359 – 366.

[27] Orna Raz. Validation of online artificial neural networks - an informal classification of related approaches. Technical report, NASA Ames Research Center, Moffet Field, CA, 2000.

[28] D. Del Gobbo and B. Cukic. Validating on-line neural networks. Technical report, Lane Department of Computer Science and Electrical Engineering,West Virginia University, December 2001.

[29] Institute of Software Reseach, Dynamic cell structure neural network report for the intelligent flight control system, *Technical report*. Document ID: IFC-DCSR-D002-UNCLASS-010401, January, 2001.

[30] M. Darrah, B. Taylor and S. Skias. Rule Extraction from Dynamic CEll Structure Neural Networks Used in a Safety Critical Application, *Proc. of the* 17^{th} *International Conference of the FLorida Artificial Intelligence Research Society*, September 2004.

[31] S. Yerramalla, E. Fuller, B. Cukic. Lyapunov Analysis of Neural Network Stability in an Adaptive Flight Control System, *6th Symposium on Self Stabilizing Systems (SSS-03)*, San Francisco, CA, June 2003.

[32] J. Schumann and P. Gupta. Monitoring the Performance of a neuro-adaptive Controller, in *Proc. of the 24th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*,Garching bei München, Germany, July 2004.

[33] J.A. Leonard, M.A. Kramer, and L.H. Ungar. Using radial bais functions to approximate a function and its error bounds. *IEEE Transactions on Neural Networks*, 3(4):624-627, July 1992.

[34] R. Tibshirani, Bias, variance and Prediction error for classification rule. Technical Report, Statistics Department, University of Toronto, 1996.

[35] L. Fu. Neural Networks in Computer Intelligence. *McGraw Hill*, 1994.

[36] G. E. Peterson. A foundation for neural network verification and validation. *SPIE Science of Artificial Neural Networks II*, 1966:196-207, 1993.

[37] K.J. Hunt, D. Sbabaro, R. Zbikowski and P.J. Gawthrop, Neural Networks for Control Systems- A Survey, *Automatica*, vol.28 no.6 pp.1707-1712, 1996.

[38] S. Lawrence and A.C. Tsoi and A.D. Back, Function Approximation with Neural Networks and Local Methods: Bias, Variance and Smoothness, *Australian Conference on Neural Networks*, Peter Bartlett and Anthony Burkitt and Robert Williamson, pp.16-21. 1996.

[39] S. Grossberg. Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121-134, 1976. Reprinted in Anderson and Rosenfeld, 1988.

[40] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23-63, January-March 1987.

[41] M. Markou, S. Singh. Novelty detection: a reviewpart 1: statistical approaches, *Signal Processing*, Volume 83(12) Pages: 2481 - 2497, December 2003.

[42] M. Markou, S. Singh. Novelty detection: a reviewpart 2: neural network based approaches, *Signal Processing*, Volume 83(12) Pages: 2499 - 2521, December 2003.

[43] C. M. Bishop. Novelty detection and neural network validation. *IEE Proceedings: Vision, Image and Signal Processing*, 141(4), 217-222. Special issue on applications of neural networks.

[44] S.J. Roberts, Extreme value statistics for novelty detection in biomedical signal processing, *IEE Proceedings Science, Technology & Measurement*. Vol. 147, issue 6, p363-367.

[45] S. J. Roberts, W. Penny and D. Pillot. Novelty, confidence & errors in connectionist systems. *Proceedings of IEE Colloquium on Intelligent Sensors and Fault Detection*, September 1996, 1996/261 : 10/1-10/6.

[46] S. J. Roberts. Assessing the Confidence of Classification and Prediction in Artificial Neural Networks. *Proceedings of IEE colloquium on AI methods in bio-signal analysis*, April 1996, 1996/100 : 4/1-4/6.

[47] P. Crook, S. Marsland, G. Hayes and U. Nehmzow. A tale of two filters - on-line novelty detection. *Proceedings of International Conference on Robotics and Automations (ICRA'02)*, pages 3894 - 3900, Washington D.C., 2002.

[48] S. Marsland, U. Nehmzow and J. Shapiro. A real-time novelty detector for a mobile robot. In *proceedings of EUREL European Advanced Robotics Systems Masterclass and Conference - Robotics , Salford*, 2000.

[49] A. Ypma and R.P.W. Duin - Novelty detection using Self-Organizing Maps, *Proceedings of Int. Conf. on Neural Information Processing ICONIP97*, Dunedin (New-Zealand), November 1997.

[50] S. Singh and M. Markou, Adaptive Neural Networks Framework for Novelty Detection in Scene Analysis, *IEEE Transcations on Knowledge and Data Engineering*, 2001.

[51] D.Y. Yeung and C. Chow, Parzen window network intrusion detectors, *Proc. Interna tional Conference on Pattern Recognition*, 2002.

[52] T. Cover and P. Hart, Nearest neighbor pattern classification,*IEEE Transactions in Information Theory*, vol.13, pp. 21-27, 1967.

[53] M.E. Hellman, The nearest neighbour classification with a reject option, *IEEE Transactions on Systems Science and Cybernetics*, vol. 6, no. 3, pp. 179-185, July 1970.

[54] V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[55] I. H. Witten, E. Frank. *Data Mining :Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers, 210-226, 2000.

[56] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proc. Computer Vision and Pattern Recognition97*, 1997.

[57] B. Scholkopf, R.C. Williamson, A.J. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *Neural Information Processing Systems*, pp 582-588, 2000.

[58] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, Vol. 2. Number 2. 1998.

[59] D.M.J. Tax and R.P.W. Duin, Outliers and data descriptions, *Proc. ASCI 2001, 7th Annual Conf. of the Advanced School for Computing and Imaging* (Heijen, NL, May 30-June 1), ASCI, Delft, 2001, 234-241.

[60] D.M.J. Tax and R.P.W. Duin, Support vector domain description, *Pattern Recognition Letters*, vol. 20, no. 11-13, 1999, 1191-1199.

[61] D.M.J. Tax and R.P.W. Duin, Data domain description using support vectors, *Proc. European Symposium on Artificial Neural Networks* (Bruges, April 21-23, 1999), D-Facto, Brussels, 1999, 251-257.

[62] D.M.J. Tax, "One-class Classification," *Dissertation*, ISBN: 90-75691-05-x, 2001.

[63] T. Yairi, Y. Kato and K. Hori, Fault Detection by Mining Association Rules from House-keeping Data, *Proc. of International Symposium on Artificial Intelligence, Robotics and Automation in Space (SAIRAS 2001)*, 2001.

[64] Machine Learning Repository at University of California at Irvine. `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[65] Y. Liu, T. Menzies and B. Cukic. Data sniffing - a machine learning approach monitoring online adaptive systems. In *Proceedings of International Conference on Tools with Artificial Intelligence (ICTAI'02)*, 16-22, Washington D.C., 2002.

[66] Y. Liu, T. Menzies and B. Cukic. Detecting Novelties by Mining Association Rules. Technical Report, 2003.

[67] J. Platt, *Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods, Advances in Large Margin Classifiers*, MIT Press, 1999.

[68] K.P. Bennett and C. Campbell. Support Vector Machines: Hype or Hallelujah? *SIGKDD Explorations*, 2,2, 2000, 1-13.

[69] B. Gärtner and E. Welzl, A simple sampling lemma: Analysis and applications in geometric optimization, *Discr. Comput. Geometry*, 2000.

[70] J. Balcazar, Y. Dai, and O. Watanabe Provably Fast Training Algorithms for Support Vector Machines, In *Proceedings of the first IEEE International Conference on Data Mining*, 2001.

[71] D.M.J. Tax, P. Laskov, "Online SVM Learning: From Classification To Data Description And Back," In *Proceedings of 2003 IEEE Workshop on Neural Networks for Signal Processing*, Toulouse,France, September 2003.

[72] T. Martinez, K. Schulten, Topology Representing Networks, *Neural Networks*, Vol.7, No.3, pp 507-522, 1994.

[73] J. Bruske and G. Sommers, Dynamic Cell Structures, NIPS 7, *Proc. of the 1994 Conference* (1995) 497-504.

[74] I. Ahrns, J. Bruske, G. Sommer, On-line learning with dynamic cell structures, *Proc. Int. Conf. Artificial Neural Networks*, vol.2 pp.141-146,1995.

[75] T. Kohonen, The Self-Organizing Map, *Proceedings of the IEEE*, Vol.8, No.9, pp.1464-1480, September 1990.

[76] B. Fritzke, Growing cell structures - a self-organizing network for unsupervised and supervised learning, *Neural Networks*, Vol. 7, No. 9, May 1993, 1441-1460.

[77] M.G. Perhinschi, G. Campa, M. Napolitano, M. Lando, L. Massotti, and M.L. Fravolini. A simulation tool for on-line real time parameter identification, *Proceedings of the 2002 AIAA Modeling and Simulation Conference*, Monterey, August 2002.

[78] M. Napolitano, G. Molinaro, M. Innocenti, and D. Martinelli. A complete hardware package for a fault tolerant flight control system using online learning neural networks. *IEEE Control Systems Technology*, January, 1998.

[79] M. Napolitano, C.D. Neppach and V. Casdorph. A neural network-based scheme for sensor failure detection, identification and accomodation. *AIAA Journal of Control and Dynamics*, 18(6):1280-1286, 1995.